universität wien

DISSERTATION

Titel der Dissertation

Provenance in Clouds:

Framework, Applications and Implication

Verfasser

Muhammad Imran

angestrebter akademischer Grad

Doktor der technischen Wissenschaften (Dr. techn.)

Wien, 2014

Studienkennzahl lt. Studienblatt:	A 786 881
Dissertationsgebiet lt. Studienblatt:	Informatik
Betreuer:	Univ.-Prof. Dr. Helmut Hlavacs

To my Parents...

Abstract

Clouds – one of the latest implementation of distributed computing emerged as a result of research and advancement in virtualization, networking, web services and grid computing. They imply a service oriented architecture and provide on-demand access to a shared pool of resources such as servers, storage, applications and services. The underlying architecture of Clouds is based on various tiers or layers such as *software*, *platform*, and *infrastructure* also called Cloud computing stack. These tiers target specific audiences such as consumers, developers and resource providers. Each tier contributes in the overall process of delivering resources, executing applications, providing services and storing data for the target users. Clouds are also abstract, modular and dynamic in nature. The *abstraction* helps hiding the underlying complex details from the end users and presents them as a single entity. The layered and dynamic architecture of Clouds has made it a rapidly adopting platform for computation and data science activities. However, there is a dire need to make Clouds more reliable, accountable, secure and trustworthy from different perspectives of the end users.

Provenance is metadata that describes the derivation history of any object in data or computation science. It is used as evidence to provide reliability and trustworthiness to the derived object. The existing works of provenance in Clouds are mostly focused on the application layer. However, it is not widely recognized that Clouds have their own provenance because of the dynamic and modular architecture. For instance, the provenance of infrastructure, platform, software, client, and virtualization tiers while delivering and hosting various applications and data.

In this thesis, we examine the architecture of Clouds and provide a list of requirements for the collection of provenance. The requirements are identified from different layers while considering various characteristics such as abstraction, modularity, and scalability etc. of Clouds. In addition, low cost of provenance computation and storage is also considered. To achieve this, a provenance framework is designed and developed which addresses the list of requirements in a modular, independent and seamless fashion for the collection of provenance. The proposed provenance framework not only addresses the identified requirements but also provides services such as storage, query and visualization of provenance.

The collected provenance and the services of framework are further utilized to present the usefulness of provenance enabled Clouds. This is achieved through validating various applications scenarios which highlight the significance of provenance and the developed framework at various layers of Clouds for the different end users. These applications cover a broad range of domains such as: (i) metadata (subset of provenance) based search, (ii) usage reports of various users and Cloud services, (iii) finding similarity patterns and utilization of resources through analysis of various users and Cloud activities, and (iv) failure tracking. Moreover, Cloud provenance is exploited as a bonding agent to explore the connections and relationships amongst various layers. In short, an effective provenance framework is derived which addresses the layered architecture of Cloud and various applications using provenance ensure the improved management of Clouds for end users. This is accomplished while keeping the computation and storage cost of provenance marginal as evident through various evaluation in this thesis. We believe that the contribution of this thesis is relevant with current data and computation science shifting towards Clouds.

Zusammenfassung

Clouds – einer der neuesten Trends in der Entwicklung von verteilten Systemen entstand auf Grund der Forschung und neuen Entwicklungen in den Bereichen Virtualisierung, Netzwerktechnik, Web Services und Grid Computing. Clouds setzen eine Service-Orientierte Architektur voraus und erlauben bedarfsgetriebenen Zugriff auf eine gemeinsame Menge an Ressourcen wie Server, Speicher, Applikationen und Services. Die darunterliegende Architektur basiert auf verschiedenen Tiers wie zum Beispiel Software, Plattform und Infrastruktur, die in der Regel als Cloud Computing Stack bezeichnet werden. Jedes dieser Tiers trägt seinen Teil zum Funktionieren der Cloud bei, sei es durch die Übertragung von Ressourcen, die Ausführung von Programmen, oder die Speicherung von Daten für die Benutzer. Clouds sind abstrakt, modular und dynamisch aufgebaut. Die Abstraktion hilft die Komplexität vor dem Benutzer zu verstecken und hilft dabei die Cloud dem Benutzer als eine einzelne Entität darzustellen. Aus Sicht der Endbenutzer haben Clouds trotz vieler Vorzüge noch einige Probleme. So sind zum Beispiel weitere Verbesserungen in den Bereichen Verfügbarkeit, Nachvollziehbarkeit, Sicherheit und Vertrauenswürdigkeit nötig.

Provenance sind Metadaten über die Herkunft und Geschichte eines Datenobjektes oder Berechnungsergebnis in der Simulationswissenschaft. Diese Metadaten werden verwendet um die Zuverlässigkeit und die Vertrauenswürdigkeit der Daten zu untermauern. Aktuelle Forschungsergebnisse zum Thema Provenance beschäftigen sich hauptsächlich mit der Applikationsschicht Wir sind aber der Ansicht, dass für Clouds auf Grund ihrer Dynamik und Modularität ebenfalls Provenance Daten gespeichert werden sollten. Zum Beispiel Provenance Daten für die Infrastruktur, die Plattform, die Software, den Client und Virtualisierungsumgebungen die für das Hosten der Applikationen und Übertragen der Daten zuständig sind.

In dieser Arbeit untersuchen welche Anforderungen eine Cloud erfüllen muss um die Aufzeichnung von Provenance-Daten zu ermöglichen. Wir unterteilen die Anforderungen anhand der verschiedenen Tiers unter Berücksichtigung der speziellen Cloud-Charakteristiken (wie zum Beispiel Abstraktion, Modularität, Skalierbarkeit, etc.). Wir schlagen zur Vereinfachung Aufzeichnung von Provenance-Daten ein Framework vor, das die identifizierten Anforderungen erfüllt und dabei modular, unabhängig und für den Benutzer transparent arbeitet. Das Framework sollte zusätzlich zur Aufzeichnung der Provenance Daten auch Möglichkeiten zum Speichern, Abfragen und Visualizern dieser Daten bieten.

Mit dem von uns entwickelten Framework und den damit aufgezeichneten Provenance Daten zeigen wir in der Folge den Zusatznutzen der Provenance Daten anhand einer Reihe von Anwendungsszenarios, welche die Wichtigkeit dieser Daten verdeutlichen. Diese Szenarien beinhalten ein breites Spektrum an Anwendungsgebieten, wie zum Beispiel Metadaten (eine Untermenge von Provenance-Daten), Benutzerstatistiken für verschiedene Benutzer und Cloud-Aktivitäten, sowie die Nachverfolgung von auftretenden Fehlern. Zusätzlich verwenden wir die Cloud Provenance Daten um die Verbindungen und Beziehungen der einzelnen Tiers zueinander zu untersuchen. Unser Framework wurde entwickelt um die speziellen Anforderungen an die Sammlung von Provenance Daten in der Cloud zu berücksichtigen und die Managementmöglichkeiten für den Benutzer zu verbessern. Des Weiteren konnten wir zeigen, dass die Anforderungen des Frameworks an Speicherkapazität und Rechenleistung vernachlässigbar sind.

Acknowledgments

In the name of ALLAH S.W.T., the most Merciful, Who blessed me with all what I am able to achieve in this world.

I want to express my special gratitude and thanks to my supervisor Univ.-Prof. Dr. Helmut Hlavacs, for the support, encouragement and guidance he showed me throughout the last years. His help, constructive comments, and valuable suggestions throughout my studies have contributed significantly to the success of this research. My special appreciation also goes to Dr. Karin Hummel, for her support, encouragement and guideline in the start of my research. It would have not been possible without her.

I am very thankful to my parents and siblings who were a constant source of encouragement during my whole life and specially my course of studies in Vienna. None of this would have been possible without the love and support of my family. I love you all.

I am thankful to my colleagues and staff members for their constant support and help during my research (thank you Tanja Schwind, Rudolf Hürner, Ewald Hotop, Dr. Andreas Janecek and all others). Special thanks to Dr. Andreas Janecek for proofreading parts of this thesis and providing me with valuable suggestions.

Many friends have helped me during my PhD studies and especially in the last weeks of writing this thesis. I greatly value their friendship and I appreciate their efforts. Special thanks to Dr. Fakhri alam and Sardar hussain for their constant help, motivation and critical analysis of this research.

I am thankful to Higher Education Commission of Pakistan (HEC) for providing me the opportunity to grasp and experience the research environment abroad by funding my PhD studies in Vienna, Austria.

Muhammad Imran
Vienna, Austria
July 2014

Contents

List of Figures

List of Tables

List of Listings

1. Introduction and Motivation

Provenance: In general, provenance is defined by the sources of information like data, processes and entities which are involved in producing an article. The Oxford dictionary[1] defines provenance as "the place of origin or earliest known history of something". Originally the concept of provenance in computation or data science was inspired from the field of art; where it was used for the verification, authenticity and finding the origin of any artifact. This involves information regarding the intermediate datasets and/or processes which are produced and consumed while producing the final product in computation or data science. There are many advantages of adding provenance to scientific computing such as verification, audit trials, reproduction, trust, reliability, and tracking to name a few [123, 31]. Provenance has been explored in various scientific domains such as grid, distributed, and workflow computing [123, 25]. Provenance has also important implications in the field of databases and Service Oriented Architectures [126, 93].

Provenance has been used in a real lab environment, i.e., in fields of chemistry and biology. Consider a researcher who is working on his experiment to find a cure for a disease in a lab environment. When executing a particular experiment, he records various characteristics and parameters in his notebook for each step during the overall experiment. This data (provenance) is later used as a proof to verify and/or reproduce the final results. The improvement in computer technology results in the architecture where the same experiments are performed using the software (in-silico) [122]. This results in the provenance collection scheme which is called automatic provenance collection; because, it is impossible to manually calculate the provenance data for such a fast and dynamic process in computational environments. In automatic provenance collection, provenance helps the scientists to trace their data which went through many modification, transformation, and interpretation.

Cloud computing: Cloud computing has emerged as a model for enabling convenient, on-demand network access to a shared resource pool of configurable elements such as networks, servers, storage, applications, and services. Historically the concept of Cloud computing can be attributed to John McCarthy[2] who said,

[1] http://oxforddictionaries.com/definition/english/provenance

[2] John McCarthy, speaking at the MIT Centennial in 1961, "Architects of the Information Society,

"If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility. The computer utility could become the basis of a new and important industry". The development and standardization such as Quality of Service (QoS) and reliability from existing fields of grid and distributed computing, Service Oriented Architectures (SOA) and virtualization contributed in the evolutionary process toward Clouds [79, 38, 61, 44, 13].

Cloud is perceived differently by various research communities and businesses depending on its domain, architecture, context and characteristics. For example, Foster et. al. [61] defines Cloud computing as "a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the internet". On the other hand Gartner [48] sees it as "a style of computing where massively scalable IT-enabled capabilities are delivered 'as a service' to external customers using internet technologies". Similarly, Berkeley RAD Lab (UC Berkeley Reliable Adaptive Distributed Systems Laboratory) describes the vision of Clouds as "Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The datacenter hardware and software is what we will call a Cloud" [22]. All the above definitions address and bring together the different aspects of Cloud computing.

As a practical baseline for this thesis, we cite [92] the National Institute of Standards and Technology (NIST)[3] definition of Cloud computing: "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". Cloud computing is divided into various components which are based on the perspective of service models, deployments architectures and the characteristics of a Cloud [92, 22]. In general, the service models are called *infrastructure*, *platform* and *software*. The deployment models mainly consist of *private*, *public*, *community* and *hybrid* Clouds. Some of the various characteristics are defined as *on-demand self-service*, *pay-as-you-go* model, *rapid elasticity*, *abstraction* of resources, and *remote* access.

Provenance of Clouds: One of the key service models, i.e., the *infrastructure* model of a Cloud provides on-demand access to various computational, and storage

Thirty-Five Years of the Laboratory for Computer Science at MIT," Edited by Hal Abelson

[3] "The NIST Definition of Cloud Computing". National Institute of Standards and Technology. Retrieved 24 July 2013

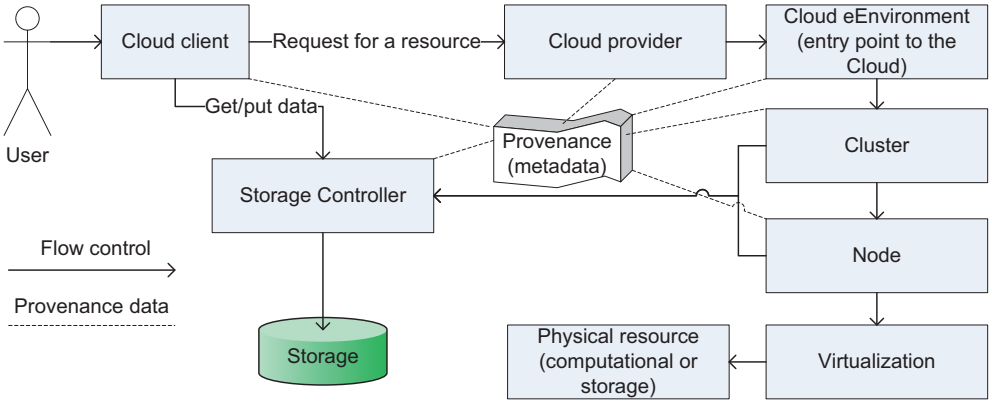


Figure 1.1: The basic flow (solid lines) of requesting a raw resource from the Cloud infrastructure. The dotted lines present the various metadata (provenance) of each step.

resources to end users. There are many steps involved while hiring a particular resource of compute or storage from a Cloud infrastructure as shown by the solid lines in Figure 1.1. However, the abstraction property of a Cloud hides these complex details from the end user's perspective. A user can hire a resource when and where it is required and once the resource is no more needed, it is simply released to the pool of resources. Similarly, the storage module of a Cloud infrastructure provides the distributed storage management for persistent data, e.g., audio, video and text files. A user can stream in/out any data by directly communicating with the storage service anytime and anywhere.

As depicted in Figure 1.1, the infrastructure of a Cloud consists of many components such as Cloud (the entry point), Cluster, Node and Storage Controller. The request of a user is routed from one component to another and each of the steps add some additional information (e.g., metadata) to the previous step for further execution as shown by the dotted lines in Figure 1.1. This information (metadata from each component) which describes the derivation process is called *provenance*. Each resource itself has a set of particular properties such as *type* and *location* for example. These basic properties are further enhanced by the installation of particular software, applications and services. Similarly, the requested resource also has a configuration property about the memory, disk space, and number of allotted CPUs etc. All these properties regarding the resources, location, time, virtual machines and the flow of request adds to the provenance data.

The *platform* model of a Cloud provides the functionality to ease the development, management and designing of complex applications such as workflows [51]. Similarly, the *software* model exposes various functions of an application in Clouds, e.g., mail services. Both the service models, i.e., the *platform* and *software* request

for various resources from a Cloud infrastructure. Communication with the Cloud infrastructure is established by a set of Application Programming Interfaces (APIs) directly by a user, via platform and/or software. Each of these models contribute to the overall provenance data because the three service models namely the *infrastructure*, *platform* and *software* interactively work as layers on top of each other. *Infrastructure* sits at the bottom, *platform* in the middle and *software* on top in a Cloud computing stack.

Concluding remarks: The evolution of Cloud from the existing computing paradigms results in added or changed features as described in [61], e.g., abstraction which hides the underlying complexities of infrastructure, platform and software. The main characteristics of Cloud computing is its total dependence on the *SOA architecture*, *modularity*, *abstraction*, *scalability* and *elasticity* etc. However, these characteristics present major challenges for the automatic provenance collection in Clouds. Similarly, the diversity of Cloud computing such as services and deployment models further enhances the challenge for the provenance collection. For instance, the service models address different view points such as users, developers and resource providers in a Cloud. These view points must be taken into consideration when providing provenance in Clouds. Moreover, the transformation for applications from grid and distributed computing towards Cloud is also in progress [69, 68]. The evolving Cloud technology and the recent trend in deploying application into Clouds results for the motivation to include the automatic provenance collection with the view point of a user, developer and resource provider.

1.1 Motivation

In Figure 1.2, we highlight different service models in Clouds such as software, platform and infrastructure which are various layers of Cloud computing. In this example, a user requests for the current weather information using the client application. All the inner details of the execution, i.e., the web services, datasets and computational resources that are used to produce the final result of the current weather are completely hidden. In a research environment, scientists are interested in the inner details of a process, i.e., step by step information regarding the intermediate processes and datasets for the final output. For example, if multiple requests for the current weather produce different results, the actual reason could be the usage of a different web service, data set, and/or infrastructure for the computation of the final result. Such information is very important in research environments in order to verify the final output, to reproduce the exact results and to bring trust and authenticity for the overall process.

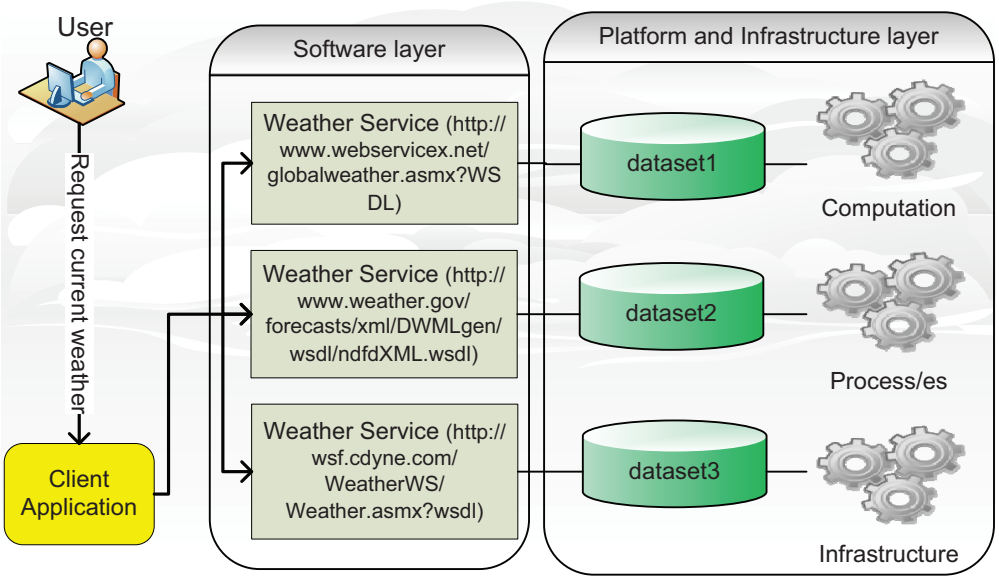


Figure 1.2: A sample application which provides information of the current weather using Cloud

The division of Clouds into various types' results in different points of view of a provider, developer and consumer called end-users. The provider supplies various computational and storage resources (infrastructure), the developer utilizes the Clouds platform for the delivery of complex and Cloud aware applications, and the consumer uses the software layer for various purposes. For example, in Figure 1.2, a user of the application is interested to know the reason for getting different results of the current weather with the same input parameters. The answer to this question lies in the various web services and different data sets used by the software application. The developer of a Cloud is interested in the platform layer where he manages various applications. This covers the deployment phases with various web services, their architectures, different versions of services and the communication model between the software and platform layers. Similarly, the real data is stored and the process is executed by physical and virtual resources supplied by a provider on the infrastructure layer. Therefore, it is important to realize that provenance in Clouds is not only associated with the application layer.

The platform and infrastructure layers contribute significant and important provenance data such as, (i) details about resources provider, the resources location and types, (ii) instance types which are based on memory, disk space and CPU cores, (iii) details about different versions of services and their communication mechanisms on the platform layer, (iv) details about changes made to various services and components of an application, (v) details about virtualization techniques and

physical resources, (vi) provenance of client applications such as web browsers, and (vii) details about various users and their group information among others. All of this information can be used in data or computation science to define various applications which address different view points of a consumer, resource provider and software developer. Moreover, the information helps in understanding the layered architecture of Clouds such as the linking and communication mechanisms between layers.

The provenance information can be utilized to make Clouds more beneficial from the end users perspective such as provenance based search, resource utilization, similarity patterns, fault tracking and management etc. The resources in Clouds can be effectively utilized by provisioning and scheduling techniques which use the provenance data. For instance, upcoming requests can be predicted based on the similarity patterns and recent requests in provenance. Provenance can also prove its significance in fault tracking and management by providing the contextual information regarding any fault and/or error. Energy consumption via the reuse of existing resources based on similarity patterns in applications and/or users is another important aspect. Provenance has many advantages and has proved its significance in distributed and workflow computing. There is a strong need to implement the provenance architecture for the evolving Cloud computing and its various types. The objective of this study is to provide provenance for Clouds paradigm along with various applications that utilize the collected provenance to make Clouds more beneficial.

1.2 Research Questions

In this study, we aim to achieve the overall goal, i.e., *provenance in Cloud computing: Framework, Applications and Implication*. This results in the following research questions which are related to the different phases of this study.

- ***Prerequisites:*** *What are the main requirements for the collection of provenance in Cloud computing?*
  Various challenges like abstraction, layering, and scalability etc. are confronted for the collection of provenance data by the dynamic architecture of Clouds. Furthermore, a list of standards such as low cost, addressing various types of Clouds, and the independence from underlying architectures and domains must be addressed. These challenges and standards together provide a list of requirements for the collection of provenance in Cloud computing.

- ***Extend the Clouds:*** *How to extend the Clouds without altering the underly-*

ing architectures and services for provenance Collection?
Clouds are abstract and various layers are hidden from each other and also from end users. Therefore, the basic mechanism for the collection of provenance requires extending the Clouds architecture in a structured and seamless fashion. Moreover, the mechanism should address various types and layers of Cloud computing in a uniform way.

- ***Provenance framework:*** *How to provide the collection, storage and management of provenance, i.e., a framework while addressing the various requirements? Furthermore, how to integrate provenance information from various layers of Clouds computing?*
  Provenance information needs to be properly managed with various operations performed by users or services in Cloud computing. Therefore, the framework should be designed in such a way that collection, storage, and management of provenance are automatic, reliable, and cost effective. Furthermore, the framework should be able to integrate the provenance data from various types of Clouds and present a unified view of provenance information.

- ***Utilization:*** *How to make Clouds more beneficial with the usage of collected provenance and the modules of the framework?*
  The presentation of various applications which take the advantage of collected provenance and different modules of the framework is important. The applications should also address various roles like consumer, developer and resource provider in Cloud computing.

Figure 1.3 presents the connection between the core part of this thesis and various questions described above.

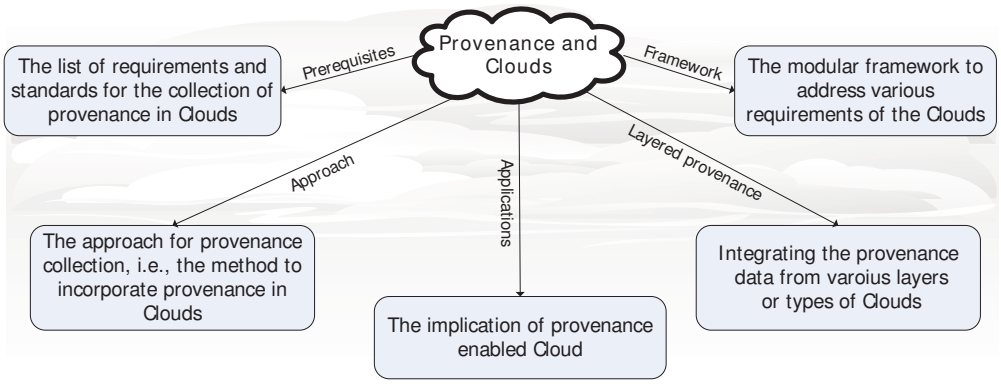


Figure 1.3: Provenance in Clouds - The overview

1.3 Research Approach

To answer the above research questions, we use the following approach:

- **Feasibility of existing provenance schemes:** We studied the existing schemes of provenance from data and computational science such as grid and workflows, and grouped them into two main categories, i.e., *provenance at core of Clouds* and *provenance is independent of Clouds*. The grouping is based on the idea if the existing schemes can be incorporated in Clouds architecture. Therefore, we present (i) the inner architecture of Clouds (specifically the research or open Clouds) which is the target domain of this thesis, (ii) categorized the existing schemes, and (iii) discussed the implication of existing schemes for Clouds with various advantages and disadvantages.

- **Finding the prerequisites of provenance in Cloud computing:** We studied the dynamic architecture of Clouds and present a taxonomy based on various service models, deployment models, roles, and characteristics of Cloud computing. We identified various requirements and list of standards that depends on the underlying architecture of Clouds. The requirements and list of standards present the prerequisites that need to be addressed while providing provenance for the Cloud paradigm.

- **Proposing a provenance framework:** We propose a cost effective and independent framework for the management of provenance data in Cloud computing. The provenance collection part of the framework adopts a technique which is not affected by the underlying architectures, domains and platforms. Therefore, the framework is available to different service models or layers, i.e., *infrastructure*, *platform*, *software*, and *storage*. Furthermore, the framework is divided into various autonomous modules such as *collection*, *storage*, *query*, *visualization*, and *user interface*. The modularity of the framework addresses various aspects of provenance utility. The objectives of the framework is to present and follow an approach where (i) minimal changes are required to the architectures and service models of Clouds, (ii) independence from underlying domains, and (iii) marginal cost for provenance collection and storage etc.

- **Extension of the framework to different layers of Cloud computing:** We present the extension of provenance collection (which is one component in the proposed framework) from one type or layer of a Cloud to others, i.e., from *infrastructure* to *platform* to *software*. We also present the important provenance information and various queries for each layer. Thus, we address

the provenance framework for the overall architecture of Cloud computing.

- **Significance of the aggregated provenance:** Each layer of the Cloud provides important provenance information. Here, we investigate the significance of the aggregated provenance from various layers and the respective view points of a consumer, developer and resource provider. The aggregated provenance is helpful to understand and explore the layered architecture of Clouds and connections between the layers. This is achieved using a sample *DataSync* application which is developed and explored from the different view points or audiences.

- **Implication of the provenance framework:** We present various applications which are designed to use the collected provenance and modules of the framework. These applications are explored from the view point of a consumer, developer and recourse provider. The applications include features of resource utilization, fault tracking, content based searching, report generation and finding similarity patterns in Clouds. These applications specify the implication of provenance aware Clouds with various characteristics and advantages.

1.4 Outcome

Cloud computing is still an evolving technology, specially, from the perspective of application deployment, e.g., the recent trend of transforming existing applications toward Clouds. Applications in Clouds are deployed and executed on the resources that are provided by different suppliers. These resources are consumed by various audiences like research communities, business organizations, stand alone users, and/or software applications. While requesting resources from Clouds, organizations such as research groups or small businesses have their preferences. These preferences result in similarities of the usage of resources. These similarities are found in data, resource types, usage time, memory, and storage etc. Furthermore, Clouds in general and research Clouds in particular rely on the communication technology between various components using open source and third party technologies (libraries, controls etc.). These technologies are often prone to faults and errors like network failure, hardware failure and software crashes. We believe, provenance is a potential candidate to resolve such situations.

The inclusion of provenance in Cloud computing will result in an environment where such data (history data, metadata) is not only provided but can be also analyzed. The analyzed provenance data can be used to find the behavior in the usage of the resources from a provider's perspective. This behavior is a common result of

similarities that exists in a research or business environment. Provenance can also be used for fault tracking and management. For instance, in case of hardware or software failures, provenance data can be utilized to find the exact component and/or device that caused the failure (fault tracking). Since, provenance also provides the contextual information regarding any failure; therefore, it can be utilized to resolve the failure by using either a proactive or reactive approach (fault management). The usage of resources in Clouds produces similarity patterns in a research environment. These patterns can be found in provenance data for the efficient allocation and intelligent organization of resources. Similarly, Cloud storage is already utilized to archive desktop data, scientific data and web application data etc. However, most of this data is persistent and unstructured. Provenance is an important ingredient to describe the context and semantics of such unstructured data. For example, we investigate the case of searching contents in a Cloud using metadata which is a missing feature in existing Clouds storage. The outcome of a provenance enabled Cloud could be simply the verification of requested resource or more complex situations such as audit trials of an organization, and/or utilization of existing resources.

1.5 Contributions

This thesis aims to present the notion of provenance in Cloud computing. The contribution is twofold; Firstly, we present a provenance framework which addresses the various requirements for the collection and management of provenance in Cloud computing. Since, Clouds are usually hardly extensible (i.e., modifications to the existing service models); therefore, we adopted a strategy to provide a provenance framework which requires minimal or no changes to the existing service models. Thus, the proposed framework is independent, generic, and not restricted to a particular domain or service models of a Cloud. This part of the thesis also addresses the issues of provenance integrity and overhead of provenance collection and storage.

The proposed framework further supports the query and visualization of provenance data which requires individual and/or the aggregated provenance of Clouds. Secondly, we explore the implication of provenance in Clouds by presenting various applications which utilizes the collected provenance data. These applications are from the view point of a user, resource provider and software developer, e.g., metadata based search in Clouds object storage, fault tracking of Cloud services and components, finding the behavior of various users, report generation of the usage of resources, and the utilization of resources by finding similarity patterns etc.

1.6 Research Publications

During this research, we published one journal paper, five international conference papers and one short paper. Our paper titled "Provenance in the Cloud: Why and How?" won the 'Best Paper Award' at Third International Conference on Cloud Computing, Grids, and Virtualization (Cloud Computing 2012). The following list provides the publications (most recent comes first) that are resulted from our research:

- "Searching in Cloud Object Storage by Using a Metadata Model", Imran, Muhammad and Hlavacs, Helmut In: The 9th International Conference on Semantics, Knowledge and Grids (SKG2013).
- "Layering of the Provenance Data for Cloud Computing", Imran, Muhammad and Hlavacs, Helmut, In: 8th International Conference, GPC 2013 and Colocated Workshops, Seoul, Korea, May 9-11, 2013, Grid and Pervasive Computing, pp 48-58, Lecture Notes in Computer Science, Springer Berlin Heidelberg.
- "Provenance Framework for the Cloud Infrastructure: Why and How?", Imran, Muhammad and Hlavacs, Helmut In: International Journal On Advances in Intelligent Systems, volume 6, numbers 1 and 2, 2013.
- "Applications of Provenance Data for Cloud Infrastructure", Imran, Muhammad and Hlavacs, Helmut In: The 8th International Conference on Semantics, Knowledge and Grids (SKG2012), Pages 16-23, IEEE Computer Society.
- "Provenance Framework for the Cloud Environment (IaaS)", Imran, Muhammad and Hlavacs, Helmut In: CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization: Nice, France, 2012.
- "Provenance in the Cloud: Why and How?", Imran, Muhammad and Hlavacs, Helmut In: CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization: Nice, France. **We Won the Best Paper Award.**
- "On using provenance data to increase the reliability of ubiquitous computing environments", Imran, Muhammad and Hummel, Karin Anna In: Proceedings of the 10th International Conference on Information Integration and Web-based Applications and Services Pages 547-550 : Linz, Austria. 2008, ACM.

1.7 Organization of Thesis

Chapter 2 focuses on the metaphor of Cloud computing. This chapter presents the detailed architecture of Cloud computing, its various components and service models which include infrastructure, platform and software. Furthermore, introduction of various roles, deployment models and characteristics of Cloud computing is provided.

Chapter 3 provides the background and related work of provenance data in computation and data science such as grid, workflow and Cloud computing. In this chapter, we investigate the existing schemes of provenance and provide a discussion to include those schemes in Cloud computing with their advantages and disadvantages. We also present the reasoning to provide provenance in Clouds, i.e., the implication of provenance aware Clouds. This chapter also presents an overview of the related concepts such as SOA, REST, XML etc. in Cloud computing.

Chapter 4 details various challenges, requirements and the list of standards that must be addressed for the collection and management of provenance data in Clouds paradigm. Based on the requirements and standards, we propose the design of the framework along with various components such as collection, storage and query among others. We also provide the implementation of the framework using different Clouds.

Chapter 5 presents various applications which utilize the collected provenance data and the proposed framework to make Clouds more beneficial from the user, developer and resource provider perspectives. This chapter also provides the evaluation of the applications and the impact on different end users.

Chapter 6 presents the implication of aggregated provenance for various layers of Clouds. Various scenarios are explored where the answer to a specific problem is achieved using the aggregated provenance. Aggregated provenance itself is accomplished via the integration of framework to multiple layers of Cloud. This chapter also provides a fault tracking system based on the aggregated provenance.

Chapter 7 details the testing of the proposed framework on various service models of Cloud computing. We provide various results of the cost of computation and storage of provenance such as using different storage and query protocols involved in the proposed framework. Hereby, we provide the impact of the developed framework like independence, marginal cost of computation and storage among others.

Chapter 8 provides the summary and conclusion of this research work as well as the limitations and open issues are described.

2. The Architecture of Cloud Computing

Cloud computing is the ability to increase capacity or add capability such as storage, computation and/or networking on the go. It relies on sharing computational and storage resources over the network. The term Cloud is a metaphor for Internet and therefore, it is also referred to as "internet based computing". There are various alternative terms of Cloud computing, e.g., utility computing, autonomic computing and virtualization; because it is used as per the understandings, knowledge and requirements by different organizations, research communities and users. In general the goal of Cloud computing is to gain the capability of high performance computing by sharing distributed and diverse resources around the world. A user can interact with Clouds and acquire any kind of a resource, application and platform by simply using a web browser or a set of provided APIs. Users are not aware of the underlying complex architecture of service models, deployment models, network infrastructure, and computation and storage resources. Cloud computing provides all these facilities with *low cost*, *scalable environment*, *on-demand* computing, *pay-as-you-go* model and *high speed access* for data storage and management.

2.1 Types or Layers in Cloud Computing

The underlying architecture of resources and provisioning of such resources is divided into various components and modules in Clouds. These components depends on the deployment model of services, the deployment model of infrastructure and the characteristics provided by Cloud computing [92, 15]. There are many kind of services model that are used in research environments and in the industry [23, 92, 84]. Few of the key and well established models are called Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). Collectively they are knows as Cloud computing stack as shown in Figure 2.1. There are many other types of service models that are described and used in industry such as, Storage as a Service (STaaS) [16], Data as a Service (DaaS), Communication as a Service (CaaS) and Network as a Service (NaaS) etc. In general, all these service models which are

used in various domains such as e-Science and businesses are called Everything as a Service (XaaS) [16, 61, 82]. In this thesis, our focus is on four service models, which are SaaS, PaaS, IaaS and STaaS. The objective of this chapter is not to compare various service models with each other, but to explore the underlying architecture and the communication mechanism between the end user, resource provider and application developer with different service models.

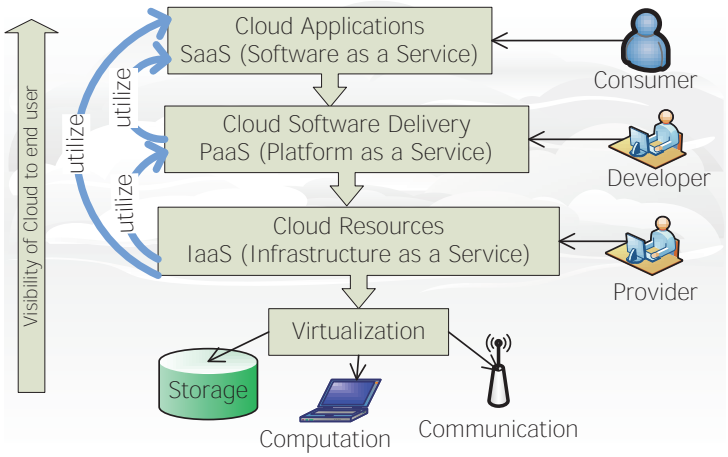


Figure 2.1: Cloud computing stack

2.1.1 Infrastructure as a Service (IaaS)

IaaS is a service model which is provided for computational, storage, servers and network resources via internet. This is considered the lowest layer in the Cloud computing stack as shown in Figure 2.1, where virtualization is used for physical resources. Elastic Cloud is a commonly used term for IaaS model and users pay for the acquired resources as they go (*pay-as-you-go*). Amazon Elastic Compute Cloud (*aws.amazon.com/ec2/*), Eucalyptus (*open.eucalyptus.com*), OpenNebula (*www. opennebula.org*) and Nimbus (*www.nimbusproject.org*) are few examples of IaaS Cloud from business and research communities. In this model, an organization outsources the equipments that are required to run various applications and/or to host any platform. The service provider is the owner and also responsible for the management of various equipments. On the other hand, a consumer or application developer has a full control of the acquired equipment and he/she can install various kinds of applications or store data. A user typically logs in to a virtualized operating system on the acquired resource; therefore, has the complete control for application deployment. A service-level agreement (SLA) is used as a contract between a consumer and provider of resources to define the delivery time and performance of IaaS

services. In the next section, we provide an example of a research or open Cloud IaaS model, i.e., *Eucalyptus* and explore the inner architecture, various modules and the communication mechanism between end user and components of Eucalyptus IaaS.

EUCALYPTUS

Eucalyptus is an open source implementation of Cloud computing IaaS service model which uses JAVA and C/C++ programming languages for various components. Users can control an entire Virtual Machine (VM) instance deployed on a physical or virtual resource [133]. It supports a modularized design and is compatible with the industry standards in Cloud, i.e., Amazon EC2 and its storage service S3[1]. 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 (being open source and by using open source technologies) which has been lacking in the business services offering such as Amazon EC2. The name Eucalyptus stands for *Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems*. Figure 2.2 presents the basic architecture of Eucalyptus Cloud and the main components are summarized as following:

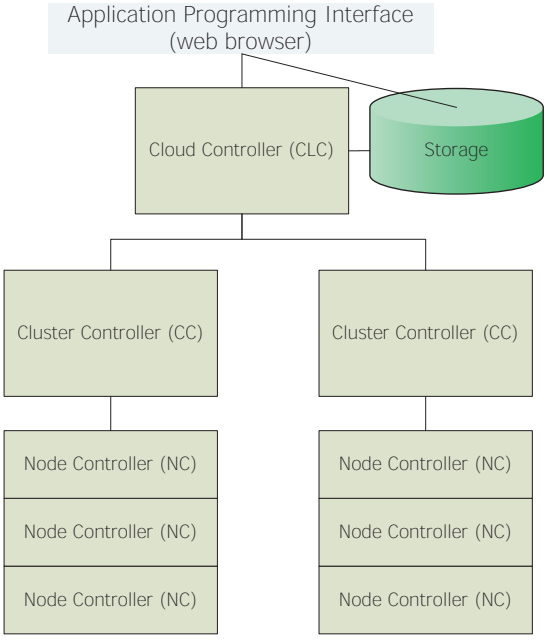


Figure 2.2: Basic architecture of Eucalyptus Cloud

- **Application Tools**: Application Programming Interface (APIs) which are available to communicate with various modules and/or services, e.g., resource

[1] http://aws.amazon.com/s3/

hiring, starting, stopping, saving and/or describing the state of a particular resource. This works as a bridge between the consumer of resources and the Cloud infrastructure. The APIs available to communicate with Eucalyptus Cloud are called *euca2ools*. Since, Eucalyptus supports the same architecture style as Amazon EC2 and S3 therefore, the support of Amazon APIs is also provided.

- **Cloud Controller (CLC)**: CLC services are the entry point for various requests from end users in Eucalyptus Cloud. The job of CLC is to process and route these requests, e.g., from consumer and administrator of a Cloud such as scheduling of Virtual Machines (creation and maintenance) by using appropriate modules, managing users data and process level agreements etc. This part of Eucalyptus Cloud is written in JAVA and the communication between a user and CLC is provided by Mule framework[2]. Mule is an Enterprise Service Bus (ESB) [80, 10, 74] that provides various interfaces for the implementation and interaction between users and CLC.

- **Cluster Controller (CC)**: A cluster is the organization of various resources (compute, storage, network, servers) that work together and can be viewed as a single system. Cluster Controller (CC) is the part of Eucalyptus Cloud that controls and manages various resources installed in a particular organization. There could be more than one cluster in Eucalyptus Cloud and the job of a particular CC is to manage various Node Controllers (explained next) that are part of a particular cluster. CC works as intermediate gateway between the Node Controller and CLC. From the architecture point of view, CC is a web service written in C language and provides a WSDL interface. From the functional point of view, CC collects the information about various Node Controllers, schedules the creation of VMs and provides the connection and configuration for the public and private IP addresses of a running VM. CC provides various functions and these functions take the request from CLC, parse the request and transform it for further processing to a particular Node Controller.

- **Node Controller (NC)**: NC is the component (service) that is installed on each physical machine. The job of NC is to administer and host the running instance of a VM. NC is written in C language and provides a WSDL interface to communicate between the VM and CC. Among various features, NC runs an instance, terminate the instance, start the network etc.

[2] http://www.mulesoft.org/

- **Storage Controller (SC)**: Cloud offers a distributed storage management *(object storage)* to archive user's data (text and pdf documents, audio and video files etc.) and raw disk images for VMs. These raw images (virtual machines) are later run as resources (operating systems, database servers etc.). Communication with a storage unit is controlled by a service, e.g., Walrus in Eucalyptus. Walrus can be used directly by users with the REST protocol to stream data in/out of a Cloud, e.g., files. Walrus can also be used indirectly by using the SOAP protocol while communicating with CLC services to upload, modify and delete virtual images.

All the communication and data exchange between different components of Eucalyptus Cloud is achieved using SOAP, XML, WSDL, REST and HTTP protocols of communication and data exchange via Axis2/C and Mule frameworks. Figure 2.3 presents the extended inner architecture of Eucalyptus Cloud along with various components, flows and the communication mechanisms that are involved.

2.1.2 Platform as a Service (PaaS)

PaaS provide tools and libraries to ease the development cost and efforts to build Cloud aware applications [52]. PaaS is an environment that allows the customer to host and manage various applications. A developer and/or ultimately the owner who is providing the application to various users has a complete control of the deployed application, e.g., various versions of the application, logic of the application and the communication protocol that are utilized etc. At the same time, developer has no control over the infrastructure of the deployed application, i.e., network, servers, and resources (computation and storage) which are provided by the supplier of a Cloud infrastructure. NIST [92] describes PaaS as: "*The capability provided to the consumer to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider.*"

There are various examples in the industry for PaaS that are famous for hosting and developing different kinds of applications. For example, Google App Engine[3] which is used to host and deploy web applications, salesforce.com [134] which is famous for CRM applications, Microsoft Azure [42] which provides infrastructure and database services through APIs (REST, HTTP and XML communication) and WSO2[4]. WSO2 is open source platform for developing enterprise applications. WSO2 provides many products such as ESB, Application Server, and Business Server among others. Similarly, the concept of providing ESB as a service is also known as

[3] developers.google.com/appengine/
[4] http://wso2.com/platforms/

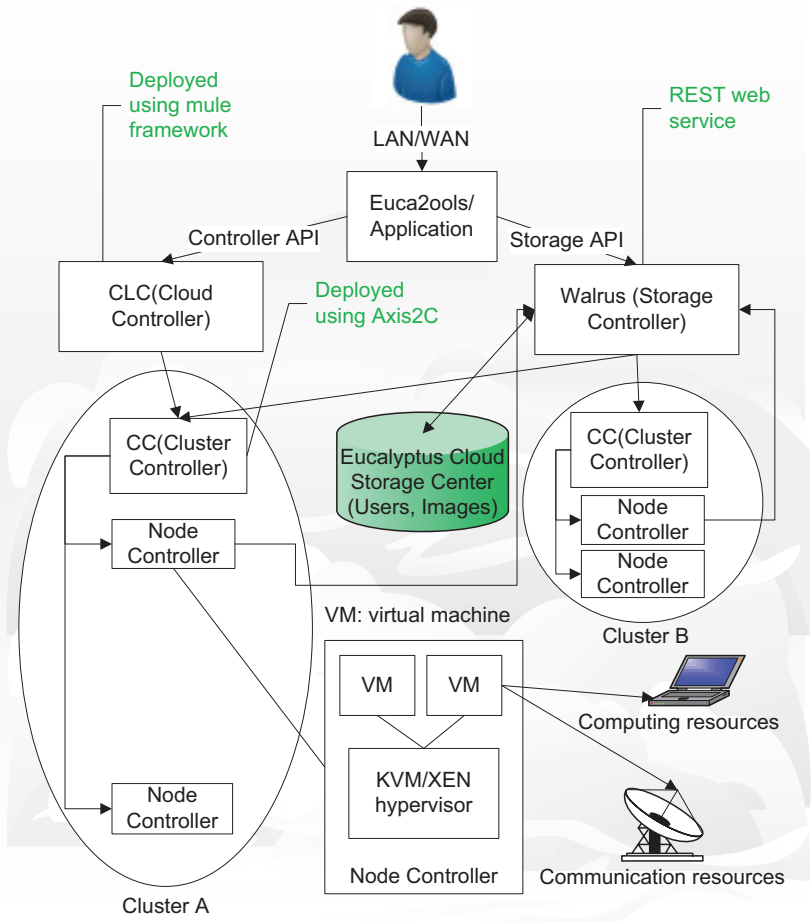


Figure 2.3: Extended architecture of Eucalyptus Cloud and various components

a type of PaaS.

In this thesis, we consider WSO2, various components of WSO2, and the underlying architecture for hosting and deploying various kinds of applications that addresses different audiences (consumer, developer, and provider) of a Cloud environment.

WSO2 Platform

WSO2 platform provides the deployment model for various kinds of applications that are using a diverse set of communication protocols (HTTP, JMS, TCP, REST etc.). WSO2 carbon is the core of the platform which provides the basic functionalities such as *logging*, *statistics*, *management* etc. It is written in JAVA and is based on the industry model called Open Services Gateway initiative (OSGI). Using the OSGI model, the entire WSO2 platform is divided into products, e.g., *Application Server*,

Business Server, *ESB* and more. All of these products (components) work together through the carbon framework, i.e., they share the run time environment of carbon core.

Furthermore, the core provides the functionality of pluggable interface. For example, when a customer requires *ESB* and *Data Server*, both of these components can be plugged in together with the carbon core. Therefore, the architecture of WSO2 follows a modular design and hereby can fit to any architectural requirement of a customer. All the components that are plugged into carbon core such as *Application Server*, *Business Server*, *Identity Server*, *Mashup Server*, and *Data Server* constitutes the WSO2 platform. The job of the WSO2 platform or any other PaaS is to eliminate the management of hardware and resources to build applications, i.e., the abstraction property of a Cloud is utilized. Figure 2.4 describes the basic architecture of WSO2 where individual layers identify WSO2 carbon core, WSO2 platform and the usage of OSGI communication protocol.

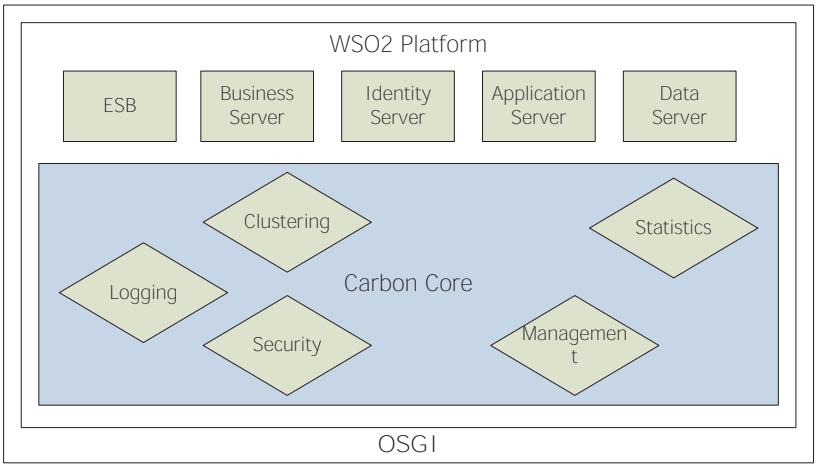


Figure 2.4: Basic architecture of WSO2 platform with Servers and Carbon Core

WSO2 Enterprise Service Bus (ESB)

WSO2 ESB is a lightweight and open source environment that allows developers to easily configure various functions such as message routing, transformation, scheduling and load balancing among others. The background technology of WSO2 ESB is the Apache Engine for exposing these various components. WSO2 ESB is developed on top of the WSO2 carbon platform and extracts the basic feature from carbon core, e.g., logging. Furthermore, WSO2 supports the functionality of add-on feature. Therefore, any component, feature or functionality can be added and removed as per the application requirements. WSO2 supports various communication proto-

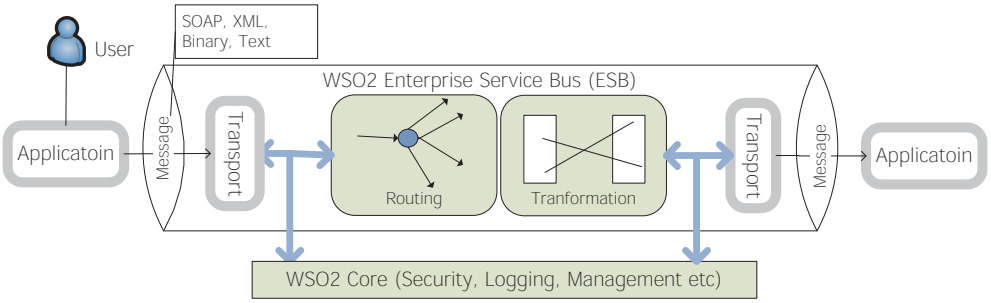


Figure 2.5: Inside architecture of WSO2 ESB

cols (HTTP, HTTPs, JMS and SMTP) between different applications and make the necessary transformation and mediation for the actual messages in different formats such as *xml* and *text* etc.

From the architecture point of view, WSO2 ESB is a software environment for middleware which enables the interoperability among various components using the SOA architecture. ESB works as a middleware mechanism for communication between the end user and complex applications by providing a single and a uniform interface. Figure 2.5 describes the inside architecture of WSO2 ESB, e.g., how a message is received and transformed from one application to another. The given Figure depicts a very basic architecture and there are many other features such as *Message Builders*, *Tasks* and *Commands* etc. that are provided by WSO2 ESB and a complete documentation can be found at[5]. In short, an ESB is described as *connecting everything to everything*.

2.1.3 Software as a Service (SaaS)

SaaS is the actual application or software that is designed, managed and delivered through PaaS and uses the resources from IaaS [59, 102]. The application is accessible from a client that is usually a web browser, e.g., gmail and/or custom Graphical User Interface (GUI). The SaaS service model eliminates the need to install the actual application on personal devices (e.g., a computer); therefore, it is very easy to maintain and upgrade any application. Different applications are centrally hosted in the Cloud and the delivery model of any application is *one to many*, i.e., a single instance of the application for each user. The user or the consumer of the application has no control over the Cloud infrastructure, e.g., storage, network and computation resources and/or the platform. The major advantage of this kind of model is the reduced cost. Users do not need to purchase the software but instead rent it (payment for the service) when and how long they require it. Various kinds of applications

[5]http://docs.wso2.org/wiki/display/ESB460/Enterprise+Service+Bus+Documentation

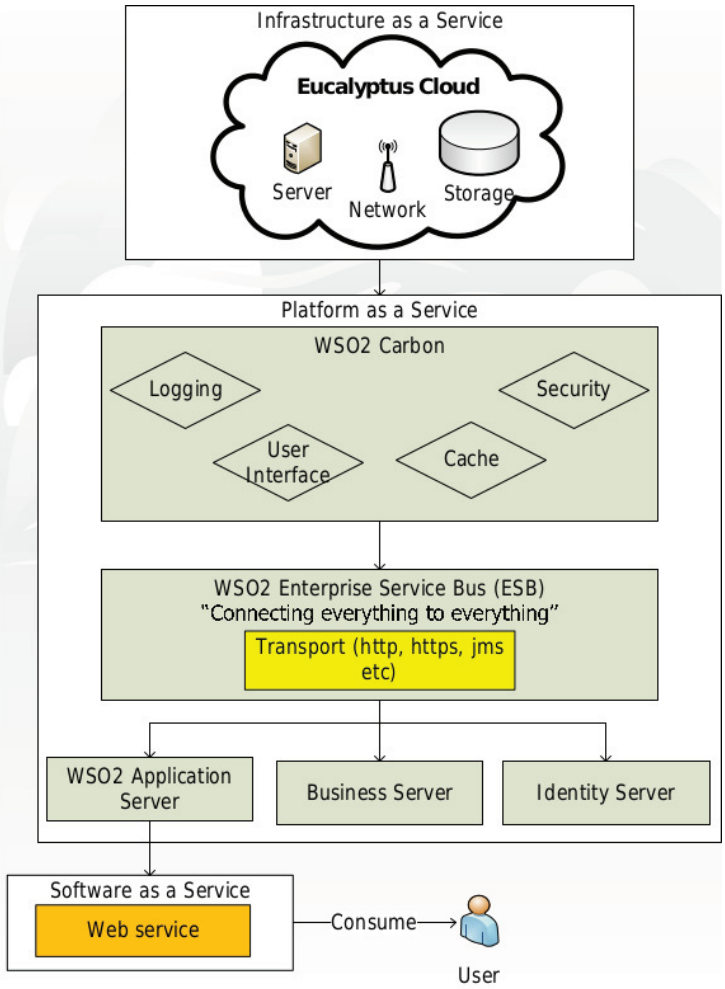


Figure 2.6: Cloud computing architecture with WSO2 and Eucalyptus IaaS

can be built and delivered as SaaS, e.g., Customer Relationship Management, Human Resource Management (HRM), Business Process Management (MBP) and web applications.

Figure 2.6 presents the complete architecture of these three deployment models (IaaS, PaaS and SaaS) of a Cloud. The resources are delivered through Eucalyptus IaaS, platform is exposed via WSO2 and the software application is a web service. It is important to note that a particular application can be developed and installed directly into Cloud infrastructure [82]. In this case, an application utilizes the resources directly from infrastructure and does not utilize any functionality from Cloud platform as shown in Figure 2.1.

2.1.4 Storage as a Service (STaaS)

Various Cloud providers such as Amazon, Google and Eucalyptus provide an independent and separate service for the management of persistent data. A consumer can upload and retrieve any data using the service and various APIs. Data can be shared with other potential users by making it public to a particular group and/or individual members. The data in Cloud is stored by following a hierarchical structure and generally it is used to store persistent data (audio, video, text files etc.). Amazon S3, Eucalyptus Walrus and Nimbus Cumulus are few examples of STaaS model. STaaS service model uses the REST protocol for the direct communication with users. This model utilizes a virtualized pool of storage resources where virtualization hides the details of the underlying file system and provides a uniform access control for the management of various data. The following is a list of tasks that can be achieved using Cloud storage:

- Archive personal data: A backup solution where users can store their data from personal devices such as computers and laptops in Clouds.
- Store applications data: To provide a fast and uniform access to various kinds of applications data. For example, an application that is used to edit various images in different formats and all the images can be stored in the Cloud storage.
- Data sharing: Any data in Cloud storage can be shared with other potential users.
- Easy to use interface: A simple GUI is provided that can be used to stream in/out any data. Furthermore, a list of APIs is available to communicate with the storage service and therefore custom GUIs can be developed.

The persistent data in Clouds is stored as objects inside the buckets. Bucket is a container (e.g., folder in file system) and the data that could be anything (audio, video, text) is represented as objects. Users can create and name various buckets and the data is stored as objects inside a particular bucket with a unique name. While storing objects, some metadata is also stored related to objects such as Access Control Policy (ACP) that contains information about the access rights, e.g., creation, deletion, modification etc. Similarly, each object has a unique identifier, i.e., a key and value attribute and this is also stored as the metadata information. It is important to note that many Cloud providers, such as, Amazon provides the facility of replicating the data, but it is missing in the open source implementation of Eucalyptus Cloud.

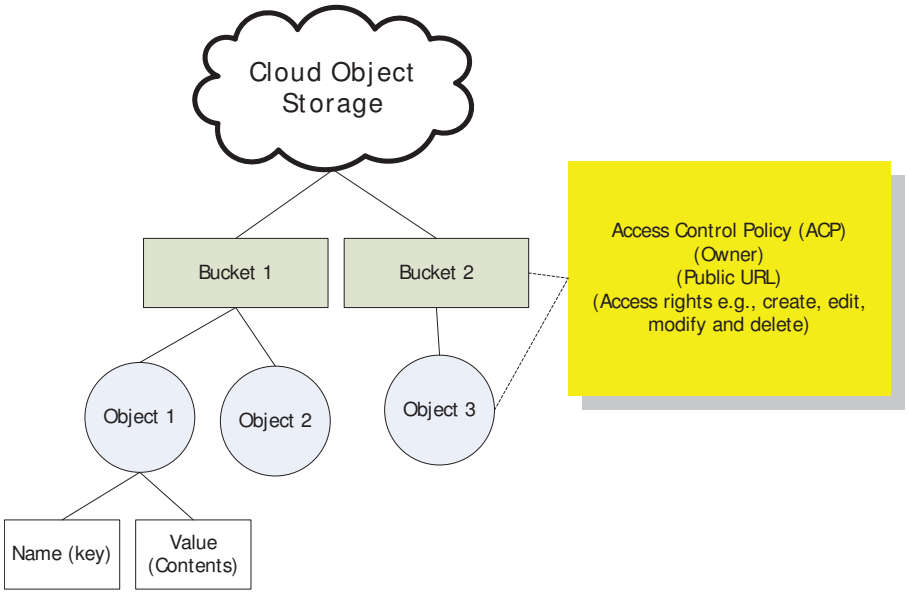


Figure 2.7: The storage architecture of Eucalyptus Walrus

The communication protocols supported by Walrus are REST, SOAP and Bit Torrent. The REST protocol is used for the direct stream in/out of data by users and the SOAP protocol is used by Eucalyptus CLC components for the management of Virtual Machine images. The Bit Torrent protocol can be used to make the data public and retrievable by simply clicking the bit torrent link. Figure 2.7 describes the basic architecture of Eucalyptus Walrus and the storage mechanism, i.e., buckets and objects along with the basic metadata information.

2.2 Features of a Cloud IaaS

The subsections below provide various features/components in a Cloud IaaS model.

2.2.1 Resource Types

Clouds host various resources according to the industry demands and user requirements. Each resource has a different ID which describes the resource type. For example, uploading multiple operating systems of Linux (Ubuntu, Debian, Solaris), Windows (XP, Vista) and database servers (SQL server, MySQL) to a Cloud IaaS result in various resources available to the users as Virtual Machines (VM). A user hires a particular resource according to the requirements of an application. A resource in the Eucalyptus Cloud is called *Eucalyptus Machine Image* (EMI). Each resource is further divided into multiple parts which are defined as Image-ID, Kernel-

ID and Ramdisk-ID. These parts detail the resource, i.e., the Kernel-ID depicts the particular version of the kernel and modules required for the proper function of an image used by a resource, the Ramdisk-ID depicts the location of the resource, and the Image-ID is used as a reference to a particular resource that is stored in Walrus.

2.2.2 Instances and their Types

A 'Cloud Instance' is a virtual machine rented from a public or private Cloud. An instance is assigned to the end user after the selection of a particular resource. Clouds offer various types of instances which are based on the memory, disk space and processor cores, etc. Eucalyptus, Amazon [1], Nimbus [101] and other Cloud IaaS divide the instances into different types such as *small*, *medium*, *large*, *extra large* and *huge* based on the hardware requirements. Table 2.1 presents these various instance types and their related memory, CPU cores and disk information from our local installation of the Eucalyptus Cloud.

Users make a request for a particular instance type based on the requirements of an application. Instance types can be changed at run time by the administrator of a Cloud to manage the load of the networked resources. A few of the properties of any instance type in Clouds are following:

- Instance-ID: It is used as an identifier for a particular instance and differentiates it from others.
- Memory: The amount of Random Access Memory (RAM) assigned to a particular instance.
- CPU: The numbers of cores assigned and the type of CPU.
- IP addresses: An instance is assigned two IP addresses called local address and public address. For instance, the local address in the Eucalyptus is used for various network modes, i.e., internal networking and the public address is used by the users to communicate with the instance.
- Owner: The user who created or started the instance and his group information.
- Storage: The amount of disk space in gigabytes (GB) that is assigned to an instance.

2.2.3 User-Data

There are three different methods to install services/applications directly on a hired resource from a Cloud infrastructure. In the first method, users make a request

Name	CPUs	Memory (MB)	Disk (GB)
m1.small	1	256	5
t1.micro	1	256	5
m1.medium	1	512	10
c1.medium	2	512	10
m1.large	2	512	10
m1.xlarge	2	1024	10
c1.xlarge	2	2048	10
m2.xlarge	2	2048	10
m3.xlarge	4	2048	15
m2.2xlarge	4	4096	30
m3.2xlarge	4	4096	30

Table 2.1: Various information of instance types from the local installation of Eucalyptus Cloud.

for a raw resource and when the resource is in running state and assigned to the user, he/she can customize the resource accordingly. This method is not feasible for the deployment of huge and complex applications because every instance needs manual changes and installation of various services, components and the networking between them. In the second method, users write their scripts which are passed to the resources as *user-data*. Such scripts are executed before the instances are assigned to the users and therefore all the services and components required by the users are already configured. This method is much more flexible and feasible for the configuration of complex and scientific applications where users get their resources ready and configured with services. In the third method, a user uploads a resource which is already configured and contains all the applications. From the resource provider perspective and with the diversity of Cloud computing, it is not possible to know all the requirements in advance for such a dynamic architecture. Furthermore, storing populated resources in Clouds require huge amount of space.

When applications are directly deployed to a Cloud IaaS, mostly custom scripts are passed to the resources which are executed after the resource is booted. The main objective of the *user-data* is to deploy services or applications on the basis of particular user requirements. For example, a custom script which would install JAVA on a requested resource before it is started is following:

sudo apt-get install JAVA (for an Ubuntu resource)

2.2.4 Elastic Block Storage (EBS)

The Elastic Block Storage (EBS) provides block level storage called volumes that can be attached to instances. The created volumes maybe formatted with a particular

file system depending on various users requirements. When a volume is created, it has a particular size (less than 10 GB), and it can be attached in the same Availability Zone (the Cluster) where the instances are running in Eucalyptus Cloud. Furthermore, multiple EBS volumes can be attached to a single instance. A volume is differentiated from others using a *Volume-ID*. Volumes are used in Clouds to store data and applications which might be require for later purposes.

2.2.5 Snapshots

To get a backup of a particular volume, a snapshot is created. A snapshot requires the name of a volume and a description can also be provided. Similar to volumes, a snapshot has a particular id called *Snapshot-ID* and contains the information of the volume from which it is created. The snapshot can be used to create further volumes or registered as a resource (VM Image) in a Cloud IaaS.

2.3 Cloud Deployment Models

An organization can choose between different options to use the resources that are provided in a Cloud environment. These different options result in various deployment models of Clouds [92]. The following is a list of few of the deployment models:

- **Private Cloud:** In a private Cloud, the computing and storage infrastructure is dedicated to a single organization. The infrastructure can be managed internally within the organization or hosted by a third party externally. In this model the infrastructure is not shared with other organizations. This kind of Cloud brings more security to the data because it is not shared across the internet. Similarly, these Clouds are more available (resources exist within the network of an organization) and fault tolerant. Private Clouds are sometimes called on-premise Clouds. This type of deployment model is best suited for organizations which are deploying critical application, using very sensitive data and/or the organization has to follow strict regulation rules etc.

- **Public Cloud:** In a public Cloud, computing and storage infrastructure is hosted by different vendors on their premises. The provided infrastructure is shared by any organization. Furthermore, the infrastructure is completely hidden and the customer has no control over the infrastructure. This kind of model utilizes *Internet as a Service* for the delivery of services, applications and storage units to end users. The cost of using these resources is paid by *pay-as-you-go* model and the resources are provided based on *on-demand* model.

This kind of Cloud has the advantage of infinite scalability and generally they cost less than private Clouds (no need to purchase expensive hardware).

- **Hybrid Cloud:** As the name suggests, the hybrid Cloud is a combination of private and public Clouds. An organization can host critical application and data on private Cloud where the applications and data that needs less security can be hosted on public Clouds. This is beneficial when an organization has sudden peaks in loads and therefore, extra load can be shifted by taking resources from public Clouds. This ensures the proper handling of increase in requirements at any time.

- **Community Cloud:** A community Cloud is controlled and used by a group of organizations that have shared interests, such as specific security requirements or a common mission. It is mostly used in universities by different research groups. The members of the community share access to the data and applications in the Cloud.

2.4 Roles in Cloud Computing

Various roles can be identified that are based on the SOA architecture and the business model of Clouds. Each of these roles, which we refer to as audiences or view points have different interests in Cloud computing. Following is a list of individual roles and their description:

- **Cloud Provider:** An organization supplying various resources for computation, storage and networking to customers or end users. These resources are accessed through dedicated APIs or a web browser. The utilization of various resources is achieved directly by a consumer, Cloud aware applications and/or various platforms.

- **Services Developer and Vendor:** The vendor provides various services and applications to customers where the provided applications utilize resources from Cloud provider. Such architecture enables the vendor to dynamically scale the application as the demand varies from time to time. For example, a new business which is not able to estimate the demand and uptake of services usage and hence the resources required, such organization can scale the application as the demand varies.

- **Cloud User or Consumer:** A user is the person or organization who is utilizing either the services offered by a vendor or resources from Cloud provider

directly. The user has no control over the deployed application or the resources infrastructure and they simply exploit the offered services or resources. For example, the storage of data in Clouds such as archiving personal data in Amazon or Dropbox by users. Similarly, in a research environment the execution of a complex application which is designed by a group of members for a particular task. Mostly the users access the Cloud environment through a thin client such as a web browser.

These roles will be used as per the above presented context for the rest of this thesis.

2.5 Characteristics of Cloud Computing

Cloud computing is defined by a set of properties and characteristics such as abstraction, elasticity and virtualization [129, 37] etc. Since, Clouds evolved from the existing domains of grid, virtualization and SOA architecture therefore, it borrows many properties from these existing fields. However, Clouds add their own specific properties to make them distinguishable from the existing computation and storage paradigms such as grid computing [61]. Here, we list the most essential properties for the understanding of Cloud computing which are used in the scope of this thesis:

- **SOA Architecture**: Cloud depends completely on Services Oriented Architectures (SOA), specially, by the usage of abstraction, modular design, and accessibility. The abstraction is achieved with the use of virtualization technology and thus exposing only the main functionalities to the end user. This makes the key functionalities of a Cloud very accessible.

- **Delivery Model (on-demand, pay-as-you-go)**: This property of a Cloud makes it different from the other fields, e.g., grid computing. Cloud offers various services and infrastructure for computation and storage resources which are managed according to different requests and demands. A resource is simply acquired when and where it is needed (on-demand). Similarly, the payment for the acquired resources is based on utility model such as electricity. Users pay for the resource as long as it is required and used (pay-as-you-go). Once the resource is no more needed, it is simply released.

- **Scalability and Elasticity**: One of the key characteristic of Clouds is the huge scalability of resources. For example, computation and storage resources can be added and removed as per the variation in demand from time to time. This is the best model to handle the requests from consumers at peak times.

This also means that Clouds are elastic, i.e., Clouds can scale in/out quickly and in automatic fashion according to the demand.

- **Miscellaneous**: There are many other properties of Clouds such as *ease of use* for the end users. The underlying complex architecture and the details are hidden from the consumer and Clouds are provisioned as a *single* and *uniform* entity, i.e., *abstraction*. For example, Cloud services and applications are accessible without the human interaction with the service provider, e.g., email. Technically, Cloud follows a *loosely-coupled* approach where the architecture is divided into many parts and components which brings *isolation* between these different components. For example, the isolation of IaaS, PaaS and SaaS layers. Similarly, there is isolation within each individual layer, e.g., various applications are delivered via PaaS and one application is not aware of the existence of any of the other applications.

2.6 Conclusion

This chapter provides the background information of Cloud computing along with various components that are related. A detailed overview is presented for various service models and deployment models of a Cloud. With each services model, we demonstrated specific examples such as Eucalyptus, WSO2 and ESB etc. Moreover, the details of the inner architecture of communication and linking of various components are presented for each services model. This chapter also provides the characteristics and various roles that exist in Cloud computing. These details will be utilized in the upcoming chapters for proposing a provenance framework and various applications that are defined by using the provenance data in Clouds.

3. Related Work

Provenance, from the French word *provenir*, "to come from" has various definitions such as:

- "Provenance is information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness." [1]
- "The history of ownership of a valued object or work of art or literature." [2]
- "The history of the ownership of an object, especially when documented or authenticated. Used of artworks, antiques, and books." [3]
- **"The place of origin or earliest known history of something."** [4]

The concept of provenance (also referred as lineage and pedigree) has been used in various fields, e.g., art and artifacts, archeology, food production, distributed, grid and workflow computing for knowing the origin and history of any object [73, 128, 65, 75]. Each of these domains define provenance within their own domain and context. Generally, provenance is used to answer the following questions in computation or data science, (i) how was a particular object (dataset) created or derived, (ii) what are the source parameters (input data) for a particular result generated by an experiment, (iii) when was a particular results generated (this may include the intermediate results), and (iv) to define the ancestry of a particular object (this can include hardware, infrastructure, platform and software components which are contributing in the overall process) etc. In this chapter, we limit ourselves to the context of computational and data science such as grid computing, workflow computing, database domain, SOA architectures and Cloud computing.

[1] http://www.w3.org/TR/2013/WD-prov-dictionary-20130312/
[2] http://www.merriam-webster.com/dictionary/provenance
[3] http://www.thefreedictionary.com/provenance
[4] http://oxforddictionaries.com/definition/english/provenance

3.1 Provenance in Grid, Workflow, SOA and Database Domains

Provenance has been a major attraction and key element in accessing the significance of electronic data [35, 117]. It is used to determine the quality and trust one can place on a final result produced by scientific experiments [87, 106]. For example, Buneman et al. [35] consider provenance as the description of origin of data and the process by which that data is produced in the field of database. Lanter [81] explains provenance or lineage as the information that describe materials and the transformations applied to those materials for the derived data in Geographic Information System (GIS).

Provenance is generally associated with the process, sub processes and datasets (e.g., input) that are consumed for the production of final data and results. Greenwood et al. [66] consider the concept of data and process together and hence define provenance as the metadata that is recorded for the process/es and data of any experimentation, e.g., workflow. Therefore, there are two important features which are considered for a final product in computation or data science.

- The data sets which are consumed and produced while delivering the final results.
- The process/es, services and components which are consumed while producing the final results.

Granularity of provenance

The amount of provenance data, i.e., granularity, which is collected for any experiment, can vary according to the application and users requirements. Provenance is mainly categorized by its granularity and there are two major approaches, i.e., *fine-grained* and *coarse-grained* provenance [116, 34]. The *coarse-grained* provenance is related with workflows that are called in-silico experiments [122]. In this case provenance data is a set of properties regarding the step by step information about process/es that were used to produce the final result. Several research works [138, 124, 118, 94] in the field of workflow provenance and scientific experimentation consider the software or the services architecture as a black box for the collection of provenance data. Therefore, the *coarse-grained* provenance scheme mainly targets the high level of abstraction in scientific applications such as the functional requirements in workflow experiments. On the other hand, *fine-grained* provenance addresses more detailed data. For instance, to record each individual data items that are used from the input datasets, e.g., tuples in the source data set which are used by a SELECT or any other query in relational databases.

There are two important characteristics of provenance, i.e., *why* and *where* which are highlighted in [35]. *Where* classifies the source elements, e.g., table and dataset and *why* justifies or give reasons for the final result. The usage of provenance in the database domain and the various techniques that are used to collect provenance data are explored in [127, 136, 28].

The execution of workflow experiments and the storage of related data require heterogeneous resources which are offered by grid and distributed computing. These include computational resources (computers, processors, and sensors), storage resources (relational database, NoSQL, file system, hard disks) and the networking of all these resources. The applications which are deployed in such an environment usually follow Service Oriented Architectures (SOA). Numerous techniques and projects have been proposed during the last few years for provenance systems in computational sciences for validation, reproduction, trust, audit trials and fault tolerance [123, 46, 35]. For instance, Provenance Aware Service Oriented Architecture (PASOA) [93] uses Service Oriented Architecture (SOA) [14] for provenance collection and its usage in distributed computing for workflow management systems. myGrid[5] and Kepler [20] are examples of projects for executing in-silico experiments developed as workflow and they use Taverna[6] and Chimera [21] schemes respectively for provenance data management. However, none of these approaches were designed specifically for the Cloud computing architecture. All the techniques and approaches which are used for provenance in grid, workflow, database and SOA mainly focus on the application or data layer of e-Science. It is important to realize that Clouds have their own provenance from the infrastructure, platform, hardware, virtualization, client and software layers.

3.2 Provenance in Cloud

Clouds are rapidly becoming the future platform for computational and data science. However, it is not realized that Clouds have their provenance in addition to the provenance of various applications [35, 117] and data [66]. This involves provenance of the infrastructure, platform, software, object storage, client, hardware and virtualization tiers among others, i.e., provenance of the dynamic and modular architecture of the Cloud itself. The exploitation of Cloud's own provenance can be used for various applications like failure tracking, privacy and security violation, reliability, accountability, usage reports of Clouds resources, and provenance based search. Provenance of Clouds to the best of our knowledge is not addressed to its

[5] http://www.mygrid.org.uk/
[6] http://www.taverna.org.uk/

full potential like the previous domains of workflow and grid computing.

The NIST [92, 84] definition of Cloud computing divides the overall architecture into many types depending on the deployment models, service models and various characteristics of Cloud. Therefore, provenance data and the target audience are different when considering these various service models, e.g., SaaS, PaaS and IaaS etc. Similarly the challenges involved in providing provenance in Clouds vary due to the variations in the deployment models, service models and other parameters such as abstraction, scalability and availability of Clouds.

The increased interest in Cloud computing attracted the research community to explore provenance in Cloud computing. The initial works to address the issue of provenance in Clouds are mostly focused on presenting the challenges associated with provenance collection and to highlight the significance of provenance data in Cloud computing [18, 99]. Specifically, the authors in [99] highlight the significance of provenance data for Cloud storage such as Amazon S3[7] and Microsoft Azure Blob[8], and they propose various applications that can be based on the suggested provenance data, e.g., content based searching in Clouds. The emphasis is to consider provenance as an important and valuable ingredient for the Cloud environment.

Clouds use a diverse and distributed set of resources for computation and storage and the networking (intranet or internet) between them that is called Cloud computing stack. These bundles of resources provide the necessary infrastructure for the delivery of a particular application and/or hosting various platforms. This architecture brings various individual layers and components that may or may not be visible to end users. For example, Clouds are accessed via web browsers and therefore the browser itself is considered as one component or layer in a Cloud environment. Daniel W. et. al. [89] consider web browsers as an important application for distributed environment and present the case of web browsers provenance. Further, they highlighted the advantages of a provenance enabled web browser such as fast web search and efficient data management when compared to a browser where provenance is not available. Similarly, virtualization is another important layer in Clouds that is achieved mostly through Xen [13] or Kernel-based Virtual Machine (KVM) [67] technologies. Provenance for Xen hyper-visor technology on the system kernel level and Cloud computing is discussed in [88]. Similarly, application level provenance, i.e., the software layer is addressed in [111] using a sample E-Social Cloud application.

The individual layers in Cloud computing and the corresponding provenance data are very important but at the same time, the aggregated provenance from various

[7] http://aws.amazon.com/s3

[8] http://www.windowsazure.com/en-us/home/features/data-management/

layers of a Cloud or any other distributed system paradigm proved to be significant. Muniswama-Reddy et. al. [95] establish the importance of integrating provenance data for different layers of workflow execution. However, their work focuses on combining the provenance data from a workflow engine, web browser and the python wrapper by extending the Provenance Aware Storage System (PASS) [96]. Similarly, a short survey [137] about various techniques from grid and distributed computing are discussed to track the data in Cloud using a layered architecture. Their work highlights the significance of provenance data at different layers of Cloud computing. Scheidegger et. al. [115] reasoned the integration of provenance data in workflow systems that include evolution, execution and specification layers of a workflow but their focus is on the application layer of a Cloud, i.e., SaaS.

Provenance Aware Storage System (PASS) [96] is one of the promising techniques successfully extended towards the Cloud environment and here we give a brief overview of the PASS project.

Provenance Aware Storage Systems (PASS)

The basic concept of PASS [98] is to observe various system calls that are made to the data from applications, e.g., READ and WRITE system calls. Since provenance of data and processes are both important to evaluate the final results, PASS also collects the relationships between data and processes. For instance, PASS creates a provenance edge (i.e., a relationship) by recording the fact that processes depend on the file, e.g., for the READ operation or system call. Similarly, when a file is written, i.e., a WRITE system call is issued; PASS also records the dependency between the processes and data.

Figure 3.1 presents the architecture of PASS for Cloud environments. PASS collects the system calls which are represented by *syscalls* in Figure 3.1 and translates these system calls into provenance records which are made from any application. PA-S3fs in Figure 3.1 is the extension of S3fs (FUSE-based file system backed by Amazon S3)[9,10], a file system interface to Amazon object storage (S3) that is utilized by PASS. Furthermore, the collected provenance data is stored in the Cloud using one of the three protocols called P1, P2 and P3 in Figure 3.1.

The first protocol P1 represented by the solid lines collects the provenance data and stores the data in the Cloud object storage (Amazon S3) together with the original data. The second protocol P2 represented by the dotted lines stores the provenance data in the Cloud database (Amazon SimpleDB) and the original data in Cloud object storage (S3). The third protocol P3 represented by the dashed lines stores the provenance data in the database (SimpleDB) and original data in

[9] http://code.google.com/p/s3fs/wiki/FuseOverAmazon
[10] http://fuse.sourceforge.net/

the object storage (S3) using a messaging service and Write Ahead Log (WAL) queue approach. Each one of these approaches has corresponding advantages and disadvantages, e.g., P1 does not provide efficient data query for provenance, because provenance data is stored together with the original data (strongly coupled).

As proposed in [88], PASS has few drawbacks. For example, PASS depends on the kernel (collector module for syscalls) and therefore any change in the kernel requires appropriate changes in PASS, e.g., reintegration. Similarly, Clouds utilize various diverse resources and to make PASS useful in Clouds will require the integration of PASS on each and individual resource (virtual machine). More importantly, PASS is designed to add provenance for Clouds storage systems. Contrary to the PASS approach we are interested in, (i) the provenance of infrastructure such as nodes, clusters and virtual machines, (ii) the provenance of platform such as composition of services and the interaction mechanisms between services and client, (iii) the provenance of software such as dataset and functional requirements of an application and, (iv) the provenance of storage in Clouds. Our focus is targeted towards the provenance of Clouds themselves.

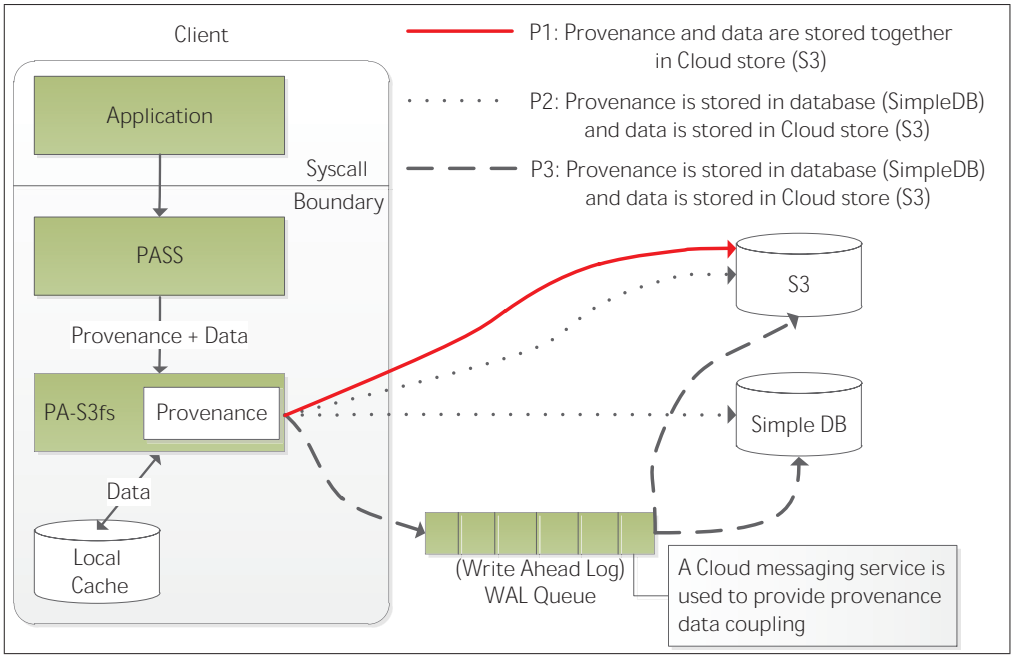


Figure 3.1: PASS model for Cloud environments: Concept taken from, "Provenance for the Cloud" Kiran-Kumar Muniswamy-Reddy, 2010

3.3 Provenance in Cloud: Why?

Cloud is an evolving paradigm which is based on virtualization, and offers different service models (Iaas, PaaS, SaaS), various characteristics such as *on-demand* computing, *pay-as-you-go* model, high *scalability*, and *abstraction* and different deployment models (private, public and community Clouds). The vision of this new paradigm is to address a complex engineering, medical or social problem [43, 103] and to store and manage huge amounts of data [19, 49] among others. Clouds enable end users to run complex applications and satisfy the need for mass computational power via resource virtualization. The execution of complex experiments towards Cloud architectures such as workflow computing is already in progress [69, 68, 85].

The existing research mainly focuses on the provenance of the application layer [91, 123, 93, 31, 76, 111]. This includes provenance of data and processes which is extremely important for quality, assurance, trust, reliability and reproducibility etc. With the layered architecture of Clouds and different roles (view points), Clouds themselves have important provenance data which adds to the provenance of applications. For example, Infrastructure as a Service (IaaS) is the paradigm of a Cloud that can be utilized by researchers to deploy complex applications as shown in Figure 3.2. This is different to existing grid and distributed environments [36, 61] where a user has to adopt his applications to the grid infrastructure and policies. An IaaS scheme provides a raw resource which is hired and updated according to the requirements of an application (by a user) without knowing the complexities and details of the underlying architecture of networking, software, services and platforms. The execution of complex applications in a Cloud requires various storage and computation resources that are hired from the infrastructure (IaaS) type of a Cloud.

There are two main approaches to deploy complex applications in Clouds. The first approach utilizes Cloud resources directly from the infrastructure [82] where the acquired resources needs to be populated with the proper tools, libraries and middleware required by that particular application. The second approach utilizes a platform layer for the deployment of applications [82]. In this case, resources are acquired by using the platform service. Therefore, the acquisition of different resources from a Cloud goes through the various components which are involved such as platform, software, storage and infrastructure. Similarly, the acquired resource has a specific type such as memory, disk space, VM image etc. Each of the components involved in acquiring a resource and the various data that specifies the acquired resources adds to the provenance data. For instance, the process of acquiring a resource goes through components such as CLC, CC, NC, Storage Controller

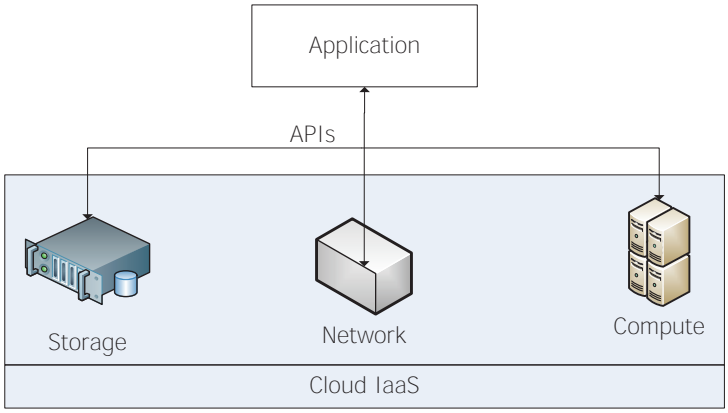


Figure 3.2: Application utilizing a Cloud IaaS

and/or Walrus in the Eucalyptus Cloud. These components route a particular request which depends on the type of the request, e.g., storing data and/or acquiring a VM. While routing a particular request to its destination, each component adds additional provenance data. For example, provenance data for the Cloud infrastructure can be broadly categorized into the categories of user's data, instances types, resources types, details about users and resource providers, memory and disk space etc.

The collected pieces of information from a Cloud IaaS are of high importance and can be utilized to define various applications. For example, a resource which is already built and updated by one user in a private Cloud environment can be used by other users with minimum or no change of the installed applications and components when the requirements are the same. Furthermore, mining provenance data can be used to forecast a future request, e.g., Eddy Caron [39] use string matching algorithms on the recent history data to forecast upcoming requests. Similarly, networks in general and Clouds in particular are prone to errors and the history data can be utilized in Clouds to resolve the faults and errors with minimum effort. For instance, the network failure in a Cloud such as the unexpected shutdown of a Node or a Cluster Controller would result in faults and errors in the deployed applications and services execution. In this case, the provenance data of the Cluster or Node Controller provides the information about the time stamp, about the particular Cluster or Node and about the method where the failure has occurred. This data can be used to pinpoint the exact failure, the time of a failure and the location of a failure. Similarly, provenance data from a software layer can be used to verify the execution of applications. Furthermore, analysis of provenance data can define various similarity patterns and behavior that exists in data and/or applications in a Cloud. The behavior can be further utilized for the efficient allocations of resources

such as predicting future requests.

In Clouds, the SOA architecture and the abstraction of networking, software, and hardware allows the end user to send and receive data to and from a Cloud without bothering about the underlying complex details. In a research environment, scientists are more interested in the overall process of execution, e.g., step by step information of a process to keep a log of sub-data and sub-processes to make their experiments believable, trust able, reproducible and to get the inside knowledge. Particularly with the improvements of in-silico experiments, most of the computation and processing is achieved using computing resources.

It can be argued that users of a Cloud may not be interested in the physical resources, e.g., brand of a computer but surely they are interested in (i) the invoked services, (ii) input and output parameters to the services, (iii) time stamps of invocation and completion, (iv) overall time used by processes, (v) methods invoked inside a particular service, and (vii) the complete process from start to finish. For Clouds this also includes the details about the provider, users, provided resources and the details of each particular resource such as type, location, size etc. hired by a user. This metadata which provides various users the ability to see a process from start to end or simply track back to find the origin of a final result including the detailed 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 (input/output parameters) for another simulation run. Provenance is still missing in Cloud environments and needs to be explored in detail as mentioned in [97, 109].

3.4 Possible Schemes of Provenance in Clouds

The existing techniques from grid, workflow and distributed computing of provenance collection and management range from tightly coupled provenance systems to loosely coupled systems [123, 46, 35]. The architecture of tightly coupled provenance collection depends on a specific domain or application because it is strongly tied with the underlying system. On the other hand, a loosely coupled provenance technique follows an independent approach for provenance collection and therefore addresses different domains, architectures and applications. Since provenance is not established to its full potential in Cloud computing, we provide a discussion of the possible schemes of incorporating provenance data and its management in Clouds with their advantages and disadvantages.

3.4.1 Provenance as a Part of Cloud Service Models (Provenance on Cloud Core)

In this scheme we assume provenance collection is part of the Cloud environment such as platform or infrastructure layer of a Cloud. For example, if we consider a Cloud infrastructure such as Eucalyptus or Amazon, the supplier of resources (Cloud providers) needs to allocate a service, e.g., provenance collection as part of the overall infrastructure. The provenance service or module has to communicate with other services of the infrastructure like Cluster, Node and Storage etc such as in a Eucalyptus IaaS. Furthermore, to store the collected provenance data, such a scheme can utilize the Cloud storage like Walrus in Eucalyptus Cloud where the original data is stored. This scheme proposes the integration of provenance as part of overall Cloud infrastructure and thus follows a tightly coupled approach. Figure 3.3 presents the architecture of such a scheme where provenance data and provenance service are part of the existing infrastructure a Cloud. Application services in Figure 3.3 represent the logic of a particular application with various web services. When a user utilizes the Cloud environment the provenance service and data are managed for the Cloud like other existing services as shown in Figure 3.3. Following are the advantages of provenance inside the Cloud infrastructure (IaaS):

- Easy to use because provenance is part of a Cloud infrastructure and the provider can decide to turn it on/off just like other services in Clouds.
- The management of provenance data (collection, storage etc.) is fast and efficient because the proposed scheme is part of the existing infrastructure and service models.
- This scheme is convenient to use because of its simplicity such as provenance collection and storage etc. are part of the existing infrastructure. Furthermore, it is the responsibility of the Cloud provider to embed such a scheme. For example, in the PaaS model, the developer can take advantage of provenance data for application developments and management without understanding the structure of provenance scheme on the infrastructure or platform layer.

The following lists the disadvantages of such a provenance scheme.

- It is unlikely that users will pay for such a scheme from the Cloud provider unless it provides benefits such as resource utilization and initialization. Furthermore, depending on the provided functionalities such as provenance data visualization etc. can further cost a consumer.

- In case of Cloud services failure, the provenance service will also fail (being part of Cloud core) and the reason of a failure cannot be traced. Similarly, in case of provenance service failure itself, there will be no records for future operations in Clouds.
- Incorporating the provenance module in different service models of a Cloud infrastructure or platform will decrease the performance of such service models. The decreased performance depends on the granularity of the provenance data and the methods to store provenance data.
- The performance of other services in Clouds is likely to decrease due to the incorporation of the provenance service as part of the existing Cloud services (infrastructure and/or platform).
- Such a scheme can only work with a particular version of the Cloud infrastructure or platform. Any change in the Cloud model or services signature needs an appropriate change in the provenance framework accordingly.
- Such a scheme is domain or application dependent and cannot be integrated in other environments.

There are various examples of provenance data management and schemes that are tightly coupled in distributed, grid and workflow computing [123, 46, 35, 116]. Each of these schemes is designed for a particular environment and they rely on the underlying service models. Therefore, these existing techniques cannot be applied to Cloud computing and the various service models of Clouds.

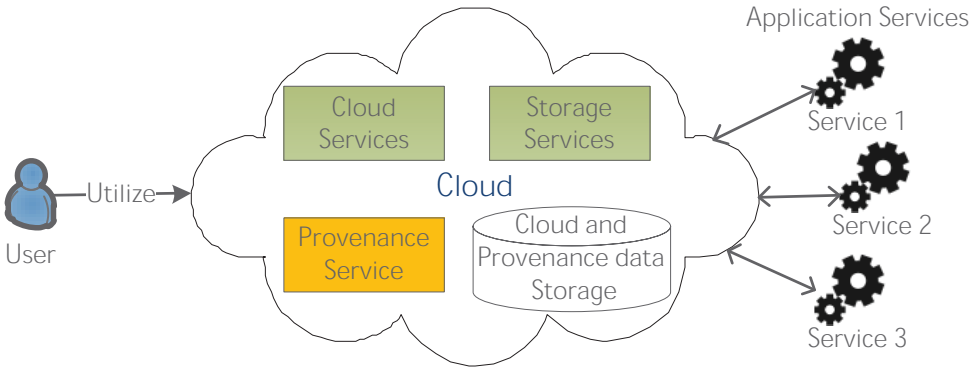


Figure 3.3: Provenance service as part of Cloud services (Cloud core)

3.4.2 Provenance is Independent of Cloud Service Models

In this scheme we assume that the architecture adopts a modular and an agent like approach to address cross platforms, applications, different domains, and various Cloud providers. This kind of scheme in Clouds is independent of the underlying infrastructure or platform. The major obstacle in implementing such a scheme is to design a technique to address properties of Clouds such as *on-demand* computing and *pay-as-you-go* model, and the extremely *scalable* and *abstract* architecture of Cloud computing. These properties of Clouds are the corresponding challenges that must be addressed when implementing an independent provenance scheme. For instance, a Node or Cluster Controller can be added at run time to the existing infrastructure in Eucalyptus Cloud. An independent provenance scheme has to address the modular design of a Cloud infrastructure to embed provenance into newly added Nodes and/or Clusters. Similarly, the inability to extend Cloud services is another major challenge for such a scheme because business Clouds are the property of various organizations such as Amazon EC2. Similarly, open source Clouds like Eucalyptus require a deep understanding of each and every service, their architecture and communication mechanisms to incorporate provenance from outside the architecture. Following are the advantages of an independent provenance scheme:

- Independent from the service models such as infrastructure and platform, and various application domains in Clouds.
- Failure of a Cloud will not affect the provenance scheme because it is not part of the Cloud environment.
- Users and Cloud providers will be able to track faults and errors if some Cloud services failed to work properly.
- This scheme is very easy to use for end users because the user has complete control over the provenance system.

Following are the disadvantages of an independent provenance scheme:

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

In workflow computing, Karma [119] is an example which adopts an independent approach towards provenance collection. Karma utilizes a notification broker where all the activities are published from workflow engine and services. The notification broker collects provenance data and stores it accordingly 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 3.4 gives a brief overview of an independent provenance scheme in Cloud. It is to be noted in Figure 3.4 that the provenance service is outside of the Cloud environment and stores the collected provenance data independent of the original data.

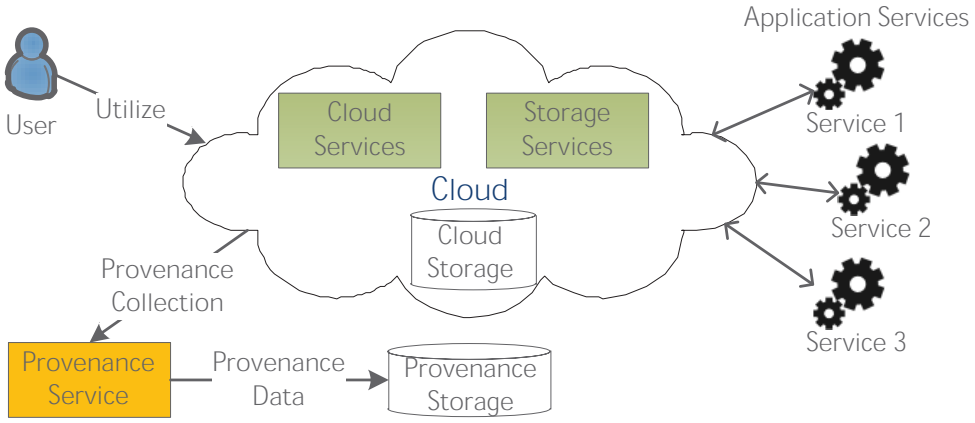


Figure 3.4: Provenance service as an independent module

3.4.3 Discussion

Both of approaches, i.e., *provenance as a part of Cloud* and *provenance independent of Cloud service models* (defined above) has their pros and cons. While considering provenance for Cloud service models such as infrastructure or platform, the major challenge is to address the extensibility of these models and services in business Clouds, e.g., Amazon. In case of open source Clouds, a developer needs a deep understanding of the functions of the Cloud and the source code in order to make or propose any change. Keeping this point in view, we propose a provenance framework in Chapter 4 which is independent of Cloud infrastructure and requires minimal or no changes in the Cloud architecture and/or service models.

Keeping in mind the existing provenance schemes from distributed and workflow computing, and the proposed implementation of those schemes in Clouds, we design and propose an approach which differs from these existing works in many ways. First, we aim to incorporate provenance in the Cloud using a seamless and modular

approach by extending Clouds in a structured way. This will bring more trust on the proposed scheme and reliability on the collected provenance data. Secondly, we aim to target the service models of Cloud computing, i.e., IaaS, PaaS and SaaS without modifying or changing the source code or the basic services architecture. The goal is to achieve provenance on the core of Cloud environment without modification and changes.

We investigated open source Clouds for infrastructure and platform such as Eucalyptus, Nimbus, Mule, WSO2 and ESB for the understanding of underlying technologies, and the communication and linking mechanisms between them. We target to intercept the communication mechanism which is used between various layers of Clouds and between different components of a single layer.

3.5 Service Oriented Architecture (SOA)

The Organization for the Advancement of Structured Information Standards (OASIS) Reference Model group defines Service Oriented Architecture [14] as: "A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations". To understand SOA, we must begin with the understanding of the term *service*. Service is a function that is well defined, self-contained, and does not depend on the context or state of other services [26]. Therefore, SOA is a collection of services that communicate with each other for various functions such as data transfer. SOA is best defined by the three patterns of publish, find and interact as described in Figure 3.5. In this model, services are published by the owner or developer to a registry. This service can be discovered by other services or users and utilized accordingly.

The concept of SOA and services is related with the web services model. The World Wide Web Consortium (W3C)[11] defines a web service as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format". Some people (from the research community) consider that SOA is different than web services. Their claim is based on the fact that SOA is not bound to any specific technology. We agree with this claim but at the same time we consider that web services are a subset of SOA model. Therefore, they are used in the same context of this thesis.

To fully realize SOA architecture, there must be a standard and uniform way of communication between services [78]. Therefore, bodies such as OASIS (www.oasis-

[11] http://www.w3.org

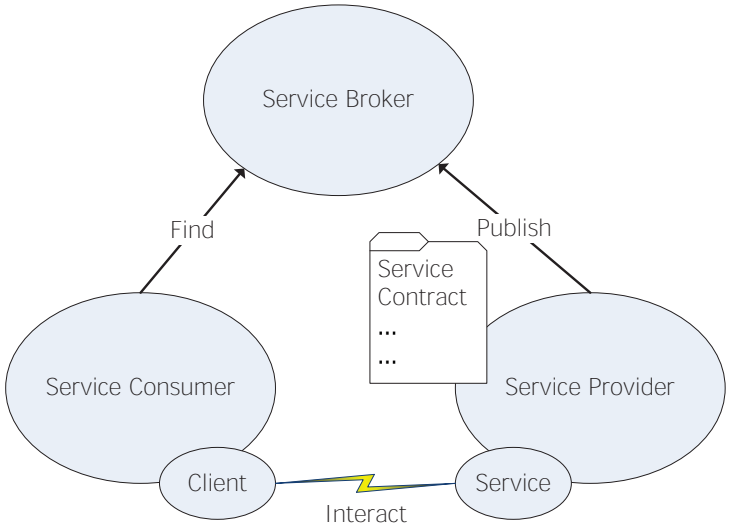


Figure 3.5: SOA model: Concept taken from W3C "http://www.w3.org/2003/Talks/0521-hh-wsa/slide5-0.html"

open.org) and W3C (www.w3.org) come into play and have defined several standards which must be met while creating SOA applications. These standards define various technologies for the communication between different layers of SOA applications such as Simple Object Access Protocol (SOAP), Web service Description Language (WSDL), Web services Resource Framework (WSRF), Representational State Transfer (REST) and others.

Clouds are designed to completely follow the SOA architecture. In the Cloud computing perspective, the SOA is considered as a paradigm that is used to build and expose an infrastructure of resources (computing, storage and their networking), a platform (WSO2, Google App Engine) for services design, development and delivery, and the execution of various applications to the target audiences (owners, developer and consumers). This architecture is used to build and deploy complex applications that require various communication protocols and huge set of resources of computation and storage.

3.6 REST

REST stands for Representational State Transfer. The architecture style of REST is a major contributor in the Web's success where it describes complex and distributed systems with beneficial properties such as scalability, performance, simplicity and reliability [104]. The basic communication model of REST is stateless where it follows a client-server model and uses the HTTP protocol. Various objects in REST

are represented by the use of unique URLs. The content of any object is accessible through HTTP GET, and to delete or modify the contents each of HTTP POST, PUT or DELETE are used. Web services which follow the REST protocol are becoming increasingly famous such as yahoo, flicker and twitter etc. The idea of REST is to utilize a simple HTTP protocol for designing complex network applications. The applications or web services that follow the REST protocol have the following advantages:

- Simple and lightweight (not a lot of extra XML markup)
- Independent of programming languages and tools such as .Net, JAVA, C Sharp etc. and independent of various platform (Windows, Mac and Linux)
- Easy to build and deploy because there is no toolkit required for the development of REST services

3.7 SOAP

SOAP, defined as Simple Object Access Protocol is an XML based protocol that is used to exchange the information between applications and/or web services in decentralized and distributed environments. Fundamentally, SOAP follows a one way messaging exchange between SOAP sender and SOAP receiver. SOAP basically uses the HTTP protocol for communication over the network however, the support for other protocols is also provided, e.g., JMS and SMTP. Just like REST, SOAP is also platform and language independent. The major disadvantage of SOAP is the use of verbose XML which can make the performance an issue. The structure of SOAP consists of the following parts:

- Envelop, Header and Body: This is the construct that defines an overall framework for expressing the contents of messages, i.e., what is inside of a message (body), who should deal with it, i.e., all or part of the message (header), and whether it is optional or mandatory.
- A data model for SOAP, i.e., a particular encoding scheme that defines a serialization mechanism that can be used to exchange instances of application-defined data types, e.g., in Remote Procedure Calls (RPC).

SOAP is a W3C recommendation and more about SOAP can be find here[12] [132].

[12] http://www.w3.org/TR/soap/

Listing 3.1: A simple document structure of XML

```
<?xml version="1.0"?>
<note>
    <to>Adam</to>
    <from>Jane</from>
    <heading>Reminder</heading>
    <body>Send me the document in pdf format</body>
</note>
```

3.8 XML

XML is acronym for eXtensible Markup Language and it is a technique of adding intelligence to the documents. XML defines a set of rules that are used for encoding documents that are both human and machine readable. XML is a tag based language and information (metadata) can be added to the tags and elements inside the tags, i.e., it is used in the descriptive sense. The communication protocols of web services such as SOAP and REST use XML data for the communication. XML is also used for the representation of data with the usage of HTML and Cascading Style Sheets (CSS) styles. XML documents are considered simple, easy to understand, portable and more powerful. XML follows a hierarchical and tree like structure as shows in the Listing 3.1, where, a note is written to Adam from Jane as a reminder. More about XML and its specification can be found here[13].

3.9 HTTP

Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems [60]. The basic model of communication is client-server where the client is usually a web browser and the server is a machine hosting a website, web service or any other application. The client submits a request message (in case of web services and SOA applications, the message is expressed as XML) to the server. The server hosting the application such as web pages or any other contents sends a response to the client which also contains additional information such as completion status etc. There are various methods which are used in HTTP and some of them are described below:

- GET: To retrieve the data from the server.
- PUT: To modify or create a resource if it does not exist already.

[13] http://www.w3.org/XML/

- POST: To consider the new contents as subordinate of the original contents, e.g., annotation of existing resources.
- DELETE: To delete specific contents or resources from the server

The above methods are sometimes referred as CRUD (Create, Read, Update, and Delete) operations. The details specification about HTTP protocol is presented in [60].

3.10 ESB

ESB stands for Enterprise Service Bus and it is a platform and architecture for the SOA paradigm. An ESB is a software architecture for middleware that provides fundamental services for more complex architectures. ESB works as an integration platform and brings together transformation and routing into a single place for various kinds of applications while providing abstraction for endpoints. Therefore, various applications communicate with each other via the bus without the knowledge of existence of other applications, and also the dependencies within a particular application.

The advantage of using an ESB is the reduced number of point-to-point communications because it is very hard to manage too many point-to-point interactions over time. Furthermore, ESB follows a simple, well defined and a *pluggable* system that scales very well for various businesses and applications. The data that travel on the bus is mostly in XML format. There are various kinds of ESBs that are provided by different companies. In this thesis we will explore two ESBs which are Mule and WSO2. Mule is utilized by Eucalyptus Cloud IaaS and WSO2 is a platform for delivering Cloud aware services. The Mule ESB is detailed in Section 4.5.1.

Various communication protocols such as HTTP, REST, SOAP, SAP and CORBA that are used by different applications are transparently translated and converted inside the ESB. Figure 3.6 presents the basic architecture of ESBs where various kinds of applications are communicating using different protocols. The basic features of ESBs such as transformation, validation and routing of messages and security of web services is also presented as part of ESBs (the core of ESB) in Figure 3.6. The detailed architecture and features, specifically from Open source ESB perspective are discussed in [72].

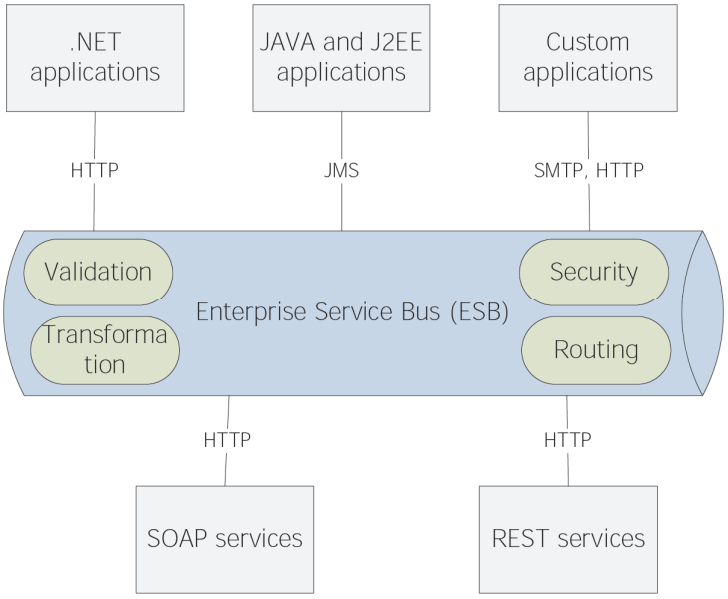


Figure 3.6: The basic architecture of an ESB

3.11 Conclusion

This chapter provides the related work of provenance from different fields of e-Science such as grid, workflow, database and Cloud computing. We analyzed the existing schemes of provenance from these various fields and divided them into two main categories, i.e., provenance as core part of Cloud and provenance as an independent service. The analysis and categorization of existing schemes is further discussed when including provenance in Cloud computing with their advantages and disadvantages. We believe, Cloud's own provenance is significant for both the service provider and the applications in Clouds. The provenance of Clouds has many application domains such as failure tracking, verification of data and processes, audit trials of the usage of Clouds resources, and provenance bases search. This chapter also provides an overview of related concepts such as ESB, SOAP, REST and XML etc.

4. A Provenance Framework for Clouds

Clouds present a layered (service models) architecture, and individual layers provide different kind of functionalities. For instance, infrastructure layer provides computation and storage resources, platform layer provides the designing and communication environment for applications development and software layer is the interface to expose the capabilities of services. The layered architecture of Clouds and the differences of functionality of each layer requires to collect, present, and manage provenance data respectively, i.e., the provenance framework should be designed in such a manner that it satisfies the Cloud architecture.

The properties of Clouds such as abstraction, scalability and the inability to extend Cloud services themselves are major requirements for provenance collection. Furthermore, the characteristics of a provenance system such as independence from the underlying architecture, modular design, low computation and storage overhead and a uniform scheme across all the layers of Clouds are important requirements for a provenance framework. Moreover, the provenance data vary according to different layers of Clouds such as client, software, platform and infrastructure tiers. Therefore, presenting the granularity of provenance data for each layer and integrating them accordingly to explore the connections such as relationships between the layers is also important. However, the existing schemes such as Karma [119], PASS [96], PASOA [93], and other [127, 136, 28, 18, 88, 98] address mostly the application layer in grid computing, e.g., workflows and/or individual layers of Clouds, e.g., data storage or software layers.

We propose a technique to collect provenance data on the communication layer (middleware) of Clouds to satisfy the requirements of layered architecture and independence from underlying domains etc. We divide our framework into many modules or components such as *collection*, *parsing*, *storage*, *query*, and *visualization* to satisfy other requirements such as modularity, low overhead and high performance etc. In this chapter, we detail the proposed framework with perspectives of provenance requirements, capabilities of the framework and the individual components (collection, storage, query, and visualization) of the framework. Following are the contributions of this chapter:

- To detail the requirements such as abstraction, high scalability, modularity, various types or layers and the inability to extend Cloud services among others that need to be addressed by the framework for the collection, storage, and presentation of provenance data in Clouds. Moreover, to define a set of standards for the provenance framework that must be achieved such as modular design, consistency and a uniform approach across all the layers of Clouds.

- To discuss the reasons to provide provenance data in Clouds, i.e., how a provenance enabled Cloud is beneficial from a user, developer and resource provider's perceptive. For instance, content based searching, efficient utilization of resources and fault tracking in Clouds utilizing the provenance data.

- To describe the mechanisms, i.e., the design of the framework for the management of provenance data while addressing the requirements and defined standards. Furthermore, to detail the architecture of the proposed provenance framework which consists of provenance *collection*, *storage*, *query* and *visualization* among others and how each component satisfy the requirements of the framework.

- To provide various provenance data and their description from different types or layers of Cloud computing.

- To detail the storage mechanisms used by the framework for the collected provenance across various layers of Cloud computing.

- To present the implementation of the proposed framework with examples from open source Clouds such as Eucalyptus.

- To present the framework experience with various advantages and disadvantages.

4.1 Prerequisites of the Provenance Framework

To provide a provenance framework that addresses the abstract, scalable and layered architecture of Clouds along with other characteristics such as modularity, independence and consistency is highly important. Furthermore, a provenance system has to provide various modules for the management of provenance data such as collection, storage and query. In the following subsections, we list the requirements and a set of standards that must be addressed to fulfill the goal of providing a provenance framework in Clouds.

4.1.1 Provenance Requirements

To provide a provenance framework for distributed architectures such as Clouds or grids, some requirements are ubiquitous like (i) collecting provenance data in a seamless fashion with a modularized design and approach, (ii) minimal overhead required for object (data items) identification in distributed environments, (iii) confidentiality and reliability of provenance data, and (iv) storing provenance data in such a way that it can be used more efficiently (energy consumption) and presenting such information to end users, i.e., query and visualization. In Clouds, we also have to address properties such as the scalable architecture, abstraction on various levels and on-demand computing along with different deployment and service models. Following is a list of requirements that must be addressed by the proposed provenance framework:

- **Domain, platform and application independence:** To provide a provenance framework that works with different domain (scientific, business, databases), platforms (Windows, Linux), applications (SOA architecture), various deployment models of a Cloud (Private, Community Clouds), and various service models (IaaS, PaaS) of Clouds.

- **Computation overhead:** Any layer of integration to solve a computer science problem brings some overhead. Therefore, the goal is to keep the computation overhead for the collection of provenance data minimal. The extra overhead of computation of provenance data required by a provenance framework can vary for a particular domain and service models of a Cloud.

- **Storage overhead:** The storage overhead of provenance data in Clouds is twofold. First, the technique to store the provenance data such as a copy of the original data is stored or a link reference to the original object is stored in provenance storage unit. This also includes the storage unit for provenance data such as SQL server, MySQL, file system (XML) or NoSQL schema. Second, the granularity of provenance data, i.e., a coarse-grained or a fine-grained approach is used for the provenance data selection.

- **Usability:** It determines the ease of use of a provenance framework from the perspectives of end users such as consumers, developers and resource providers of a Cloud. The usability constitutes on the properties such as how to activate, deactivate and embed a provenance framework into existing Cloud infrastructures and service models. Usability also identifies that the provenance framework is completely independent or modifications are required on various layers of a Cloud.

- **Object identification:** To adopt a consistent approach for the identification of various objects in Cloud computing, i.e., to link the provenance data with the original objects across various layers of Clouds. The link between original objects and provenance can utilize one of many methods such as making copy or keeping reference of the original objects.

- **Automaticity:** With the huge amount of data and process computation within a Cloud, collecting and storing provenance data should be automatic and consistent.

- **Properties of Clouds:** To provide mechanisms that addresses various properties of Clouds such as *on-demand* computing, *abstraction* and *scalability* among others. For instance, the scalable architecture of Clouds allows the addition and removal of various compute and storage services at run time. Therefore, we have to properly handle the availability and extensibility of the framework to the added/removed services.

- **Interaction with Cloud services:** Business Clouds such as Amazon are property of organizations. Therefore, service models that are offered by business organizations are not extensible, i.e., the direct interaction with various services is not possible. On the other hand, Open source Clouds such as Eucalyptus need an understanding of every service, functional requirements and source code if a change is required. For instance, the interaction with the infrastructure services for the collection of provenance data. It is extremely important to design and provide an approach to handle such situations, e.g., an independent provenance scheme which requires no or minimal change in the existing architectures of Clouds.

- **Granularity of provenance data:** The framework has to address the distributed architecture of Clouds with different service models, i.e., IaaS, PaaS, STaaS and SaaS for the collection of provenance data. The granularity of the provenance varies with each of these models. Furthermore, the collected provenance should address different roles (view points) in Cloud computing such as consumers, software developers and resource providers. Therefore, to provide a list of significant provenance data according to the architecture of Clouds is important.

- **Types of Cloud (Service Models) and flow of provenance data:** To provide techniques for the management of provenance data that can be uniformly applied to different services models or layers such as IaaS, PaaS and SaaS. Furthermore, the collected provenance data from various layers should

be stored in such a manner that the relationships between layers are exposed to end users. For instance, to manage and expose the flow of activities from top to bottom; SaaS, PaaS and IaaS layers.

4.1.2 Provenance Metric

We defined a set of parameters that should be addressed while providing a provenance framework to address the issue of incorporating provenance in Clouds such as various service models. These set of parameters include the following:

- **Independence from Clouds architecture:** The proposed framework should be independent of the underlying architectures, various components and service models of a Cloud.
- **Consistency:** The framework should be consistent across the layers of Cloud computing. Therefore, the extension of the framework from one layer to others should require minimal or no changes.
- **Marginal cost:** The framework should adopt policies to keep the computation and storage overhead minimal to affordable.
- **Data relation:** To keep the relationships that exists within provenance (in-line) and between provenance and original objects (out-line). We intent to maintain the in-line relationships using XML structure which tie parent and child elements. Similarly, the out-line relationship is maintained in XML using a reference or link to the original objects.
- **Efficient query:** To provide the support of querying the provenance data to extract important information based on users retirements. Various queries can be generated that depends on the underlying service models, e.g., to query the usage of Nodes and Clusters in a Cloud IaaS. It is also important to provide the support of queries which require aggregated provenance from various layers of Cloud computing.
- **Modularity:** To address the various requirements offered by a Cloud paradigm for the provenance framework in a seamless, automatic and modular fashion.

4.2 The Design of the Provenance Framework

In this section, we outline our provenance framework which addresses the requirements, various provenance data and the defined metric in a seamless and modular fashion. As discussed in Section 3.4, one of the methods (*provenance as core part*

of a Cloud) to implement provenance in Clouds (e.g., IaaS model) is changing the source code. Such a method is motivated from research works [123, 46, 35] in distributed and workflow computing for provenance data management. This is very cumbersome because deep understanding of the source code and functional requirements is required. Therefore, such a method will restrict the changes to a particular version of a Cloud. Moreover, this method is not feasible to address and satisfy the provenance requirements for various Cloud providers, domains and applications.

The other method (*provenance as an independent module*) follows somehow an independent approach such as Karma [119] where the provenance module works as a bridge between the provenance store and Cloud services. This scheme is not possible for business Clouds because the service models are not extensible. Furthermore, this scheme requires the interaction between Cloud services and provenance module such as request, permission and response messages which might increase the cost of the framework and thus affect the performance of Clouds significantly. To overcome these problems we propose an approach for the collecting of provenance data on the communication and linking (middleware) layer of Clouds.

The architecture of research Clouds rely on open source third party tools, libraries and applications to link various components and services. For instance, Eucalyptus Cloud depends on the Apache Axis, Axis2/C, and Mule frameworks. These third party libraries are used for the communication and linking mechanisms between various components of Cloud computing. The architecture of Clouds is the orchestration of different services and the third party libraries works as middlewares to connect those services.

Middlewares are components in different service models of Clouds where all the communication is taking place. We propose an approach which extends these middlewares/libraries used by Clouds. The black box design for such an extension is depicted in Figure 4.1. The extension is achieved with custom methods implemented to collect provenance data at various levels/layers of Clouds. Figure 4.2 presents the approach of the collection of provenance data on the middleware level for different service models of a Cloud. The middleware is configured with custom logic for the collection of provenance data. This approach requires minimum efforts and can be deployed across any Cloud system that uses the same middlewares/libraries. Furthermore, there will be no changes required in the architecture of services and/or signature of a Cloud because only the middleware is updated. Other services in Cloud computing are not affected by this approach as shown in the Figure 4.2.

We introduce various components of the framework as shown in Figure 4.3 to address the scalable and layered architecture of Clouds and to answer the requirements of provenance in Clouds. These various components are listed in the following:

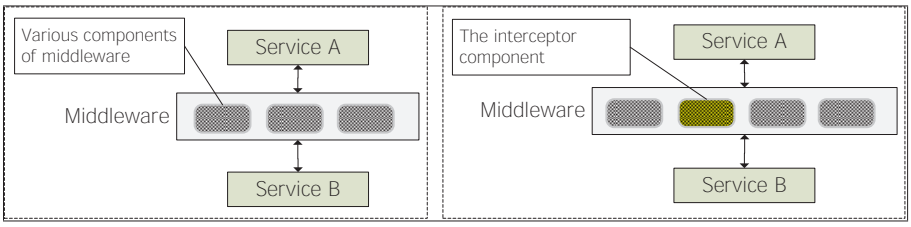


Figure 4.1: The left side depicts a middleware which connects two services. The right side presents the extension of the middleware by introducing a new component, i.e., interceptor

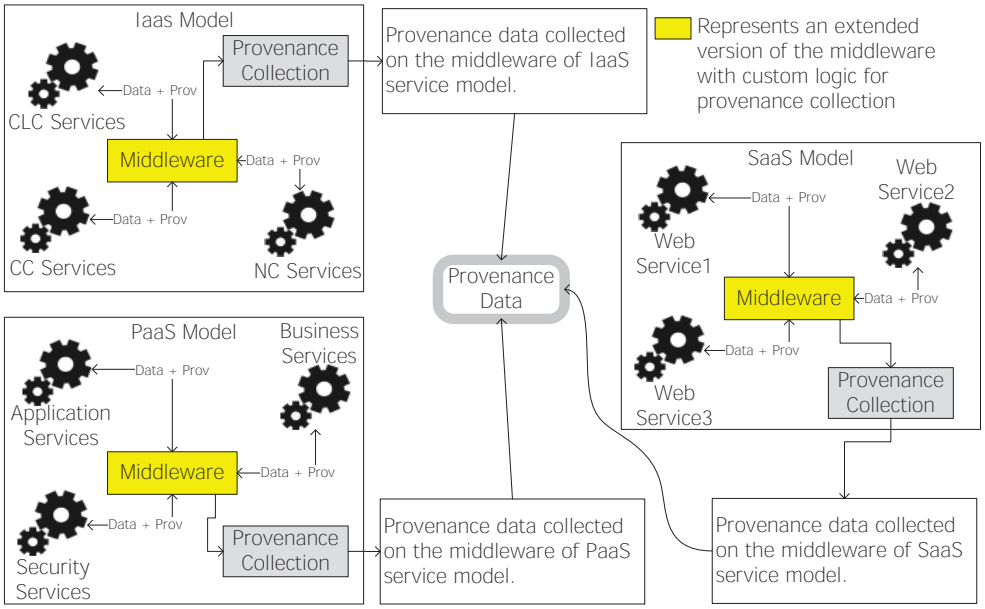


Figure 4.2: The basic approach for the collection of provenance data in different service models of Cloud computing. Provenance data is collected at the middleware of each model.

- **Provenance Collection:** Provenance data is collected by adopting a technique which extends the underlying middlewares of Cloud computing for different service models. Using this technique, the basic architecture of Clouds remains the same and no changes are made to services architecture. Since the data is collected without altering the architecture, we expect the computation overhead to be minimal.

- **Provenance Parsing:** The collected provenance data is parsed according to the layers of Clouds. For instance, IaaS, PaaS, and SaaS data is parsed using various parsers for the significant provenance data.

- **Provenance Data:** This part of the framework presents the data types, their values and description, i.e., the granularity of the provenance data for various

layers of Cloud computing.

- **Provenance Storage:** The parsed provenance data is stored in a well-defined XML schema or database storage for future utilization such as content based searching. This section also presents the techniques, i.e., object storage and database storage for provenance data to keep the storage overhead minimal and take the maximum advantages of the stored data such as efficient query and visualization.

- **Provenance Query:** This module presents the architecture of various protocols for efficient, fast and reliable results of users queries utilizing the provenance data. The protocols depend on the storage mechanisms, i.e., we used LINQ to XML for object storage and AMAZON APIs for database storage of provenance data.

- **Provenance Visualization:** The results of users queries are visualized in different formats such as charts, lines, graphs and pies etc.

- **User Interface**: An easy to use interface provided with the framework for the configuration of different components of the framework for various service models and components/services of individual models.

We provide the detail of various components of the framework in the following subsections.

4.2.1 Provenance Collection

A Middleware [27, 113] is a software that connects and integrates different components of complex applications in distributed environments. Middlewares provide support for communications and transactions between the components among others. The communication between components is taking place using a variety of protocols such as Remote Procedure Call (RPC), Java Remote Method Invocation (RMI), Java Message Service (JMS) and Web services for examples. Web services use XML format for the content of messages and SOAP or REST protocol for the communication.

Middlewares provide the functionality of interception for various purposes such as Quality of Service (QoS), versioning, security and privacy [47, 50, 105]. The interception provides a low-level mechanism to intercept the communication messages without altering the architecture. Therefore, middlewares can be extended with custom logic using interceptors. The communication between services of distributed architectures is taking place in various flows such as normal flow and faulty flow in

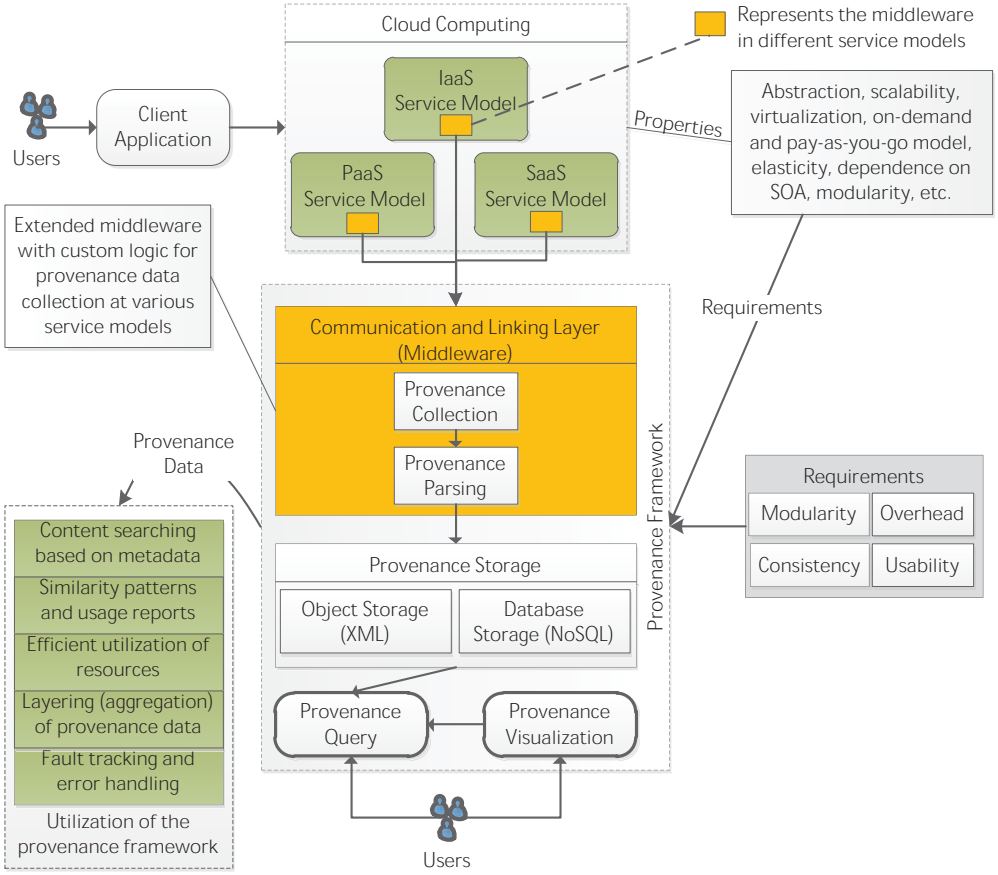


Figure 4.3: The architecture of the proposed framework with various modules and components, and their relation with the requirements.

Apache Axis [62]. The flow of communication is two ways, i.e., request and response messages.

We developed the interceptors for the collection of significant provenance data and deployed them in middlewares thus extending the architecture of Clouds. These interceptors are placed in various flows for the collection of significant provenance data. The interceptors take the hold of the request as well as response communication message as shown in Figure 4.4. Once the data is collected from SOAP and REST messages at various layers of Clouds, it is forwarded to the parser module for further processing.

4.2.2 Provenance Parsing

The parser receives SOAP or REST messages from IaaS, PaaS and SaaS layers. The parser works in-line with the collector module to avoid altering the architecture of the Cloud. The parser is divided into sub-parsers because of the layered architecture

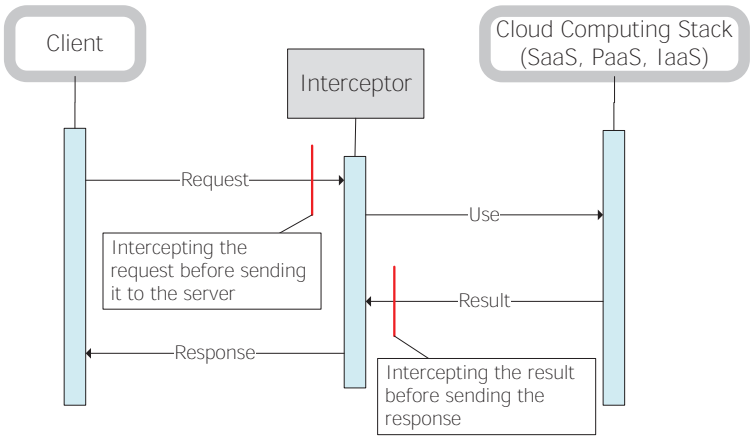


Figure 4.4: Collecting the provenance data using interceptor between a client and Cloud stack

of Clouds namely: *Infrastructure Parser*, *Platform Parser* and *Software Parser* as shown in Figure 4.5. In our development of the provenance framework, STaaS is part of the Cloud IaaS. Therefore, the *Infrastructure Parser* is further divided into two sub-parsers named: *Storage Parser* and *Computation Parser*. These parsers gather significant provenance information and neglect the rest of the message as following:

- **Infrastructure Parser:** This parser collects information for computation and storage requests on the infrastructure layers. In case of a computation request, it determines the flow, resource type, instance type, resource provider, resource user, creation time, termination time, IP addresses of VMs and user data etc. In case of a storage request, it collects information such as consumer name, group details, time used by service, data types, data size, and storage location etc.

- **Platform Parser:** This parser collects information such as commutation and linking protocols, developer name and group information, changes made to services such as different versions etc.

- **Software Parser:** This parser collects information for applications executed in Clouds such as web service names, method names, input and output parameters, and time taken by services among others.

The parsed data is represented in a well-defined *XML* structure. We use *XML* schema for the collected provenance information because it is a widely used model for data representation. Furthermore, *XML* can be used to maximize the advantages such as custom algorithms and third party applications from various users which

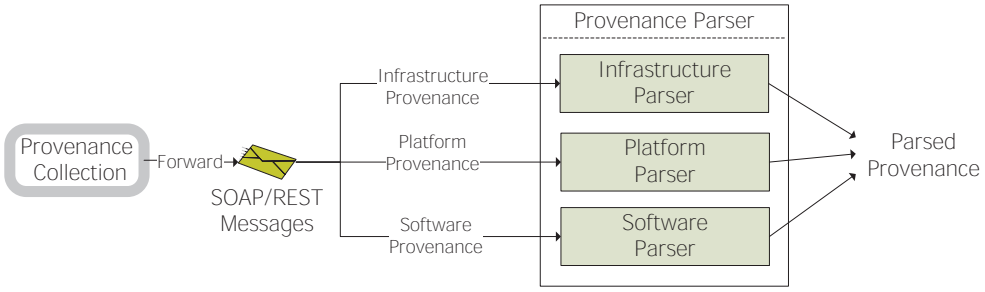


Figure 4.5: Provenance parser for different layers of Cloud computing

utilize a well formed provenance. It is also useful to provide a standard schema and hence the usage according to individuals preferences such as querying the provenance data.

Table 4.1 presents a sample of collected and parsed provenance data by the proposed provenance framework. The data in Table 4.1 represent activities of various methods of the Eucalyptus Cluster service and details the timestamps, resource type, resource location and instance specific information such as memory size and disk space. $<UserData>$ is a list of applications specified by a user to populate the resource which vary according to individuals preferences and $<TimeStamp>$ are corresponding start and finish time of a web service method in Table 4.1.

Eucalyptus Data Item	Eucalyptus Data Values
Service Name	<EucalyptusServiceName> ClusterController </EucalyptusServiceName>
Method Data	<MethodName> StartNetwork </MethodName> <StartTime> 2014-02-11:11:25 </StartTime> <FinishTime> 2014-02-11:11:26 </FinishTime> <ClusterAddress> 131.130.32.12 </ClusterAddress> <UserID> admin </UserID>
Image Data	<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 Data	<Name> m1.small </Name> <Memory> 512MB </Memory> <Cores> 1 </Cores> <Disk> 6GB </Disk>

Table 4.1: Various provenance data represented in XML elements of different methods of Eucalyptus CC service

4.2.3 Provenance Data

The parser module parses the data according to different service models. Therefore, to identify important provenance information according to various layers or service models is important. Following is a list of important provenance data that is required and collected for the Cloud infrastructure (IaaS).

1. Cloud process provenance: The control flow in a Cloud environment, i.e., the sequential execution of various services and processes is important provenance information. For instance, a request goes through CLC, CC, Walrus and NC services in Eucalyptus Cloud. Therefore data such as web service name, method name and timestamps of invocation and completion in particular are important provenance information. Similarly, the control flow of various components inside a particular service is also important. For example, the Walrus service in Eucalyptus Cloud has components such as *BukkitWS*, *BukkitInternalVM* and *WalrusReplyQueueWS* etc. Such provenance information can be utilized for various purposes, e.g., finding the exact method and time when a failure occurs. Moreover, Cloud process provenance also includes various IDs which are assigned to resources such as reservation-IDs and instance-IDs.

2. Cloud storage provenance: The data which is stored from various consumers or applications in a Cloud object or database storage has valuable provenance information such as type of data, size of data, creator of data and the storage location. Furthermore, the time information about the creation, update and deletion of data items, and the time details about sharing the data items with other members of a Cloud is significant provenance data.

3. System provenance: System information or physical resource details like compiler version, operating system, VM details, and the location of virtualized resources. For example, if a VM fails to work properly, its location can be found in provenance data.

4. Timestamps: Invocation and completion time of various Cloud services and their methods are important provenance information. For instance, the total time required to stream in/out a particular data item in Eucalyptus Walrus or the time required to hire a particular resource from a Cloud IaaS is calculated based on the invocation and completion times.

5. Consumers: The details about various users of a Cloud such as their names, group information and Access Control Policy (ACP). Such provenance is important for various tasks like security and privacy of data in Clouds.

6. Provider: Various details about the supplier of a Cloud such as the organization name, location of clusters, nodes and storage units. Such information are important because their could be laws against the usage of resources from a particular geographical area.

7. Instance provenance: A Cloud instance is a virtual machine in a running state. Instances in Cloud vary in sizes based on hardware requirements. The provenance data includes information like disk size, memory, resource type, number of cores (CPU) and the number of acquired instances from various users or applications for a particular operation.

8. Cloud User-data: This data is part of a Cloud infrastructure when applications are deployed directly on IaaS model. When users hire various resources, they populate them according to the application requirements before the resource is booted. This includes the initial scripts which contain information such as tools and libraries that are required to deploy the application. For instance, a script which downloads and installs JAVA JDK, JAVA JRE, MySQL database system, and Tomcat web services engine.

The above items present important provenance information from the infrastructure (IaaS) part of Cloud computing. Figure 4.6 presents this information in the context of Eucalyptus Cloud when a user requests for a computational or storage resource. The platform and software layers also provide significant provenance data from the view point of software developer and software consumer such as:

1. Application provenance: Information about various processes of an application, e.g., process name, method name and the time taken by a single or overall process.

2. Data provenance: Information about the input and output parameters that are passed to a particular process which include the initial data sets that are consumed, the intermediate data sets that are produced and the final result.

3. Platform provenance: Various information that are collected on the platform layer such as different versions of services and the details of modifications made to a particular service and its methods. For instance, who made the modifications, and when various changes were performed and uploaded to a Cloud.

4. Other: There are various provenance information which are similar for all service models such as the flow of execution in IaaS or SaaS model, the IP

addresses of various VMs which are requested by users and applications, and the timestamps of various invocations for web services. Such information are gathered for each service Models.

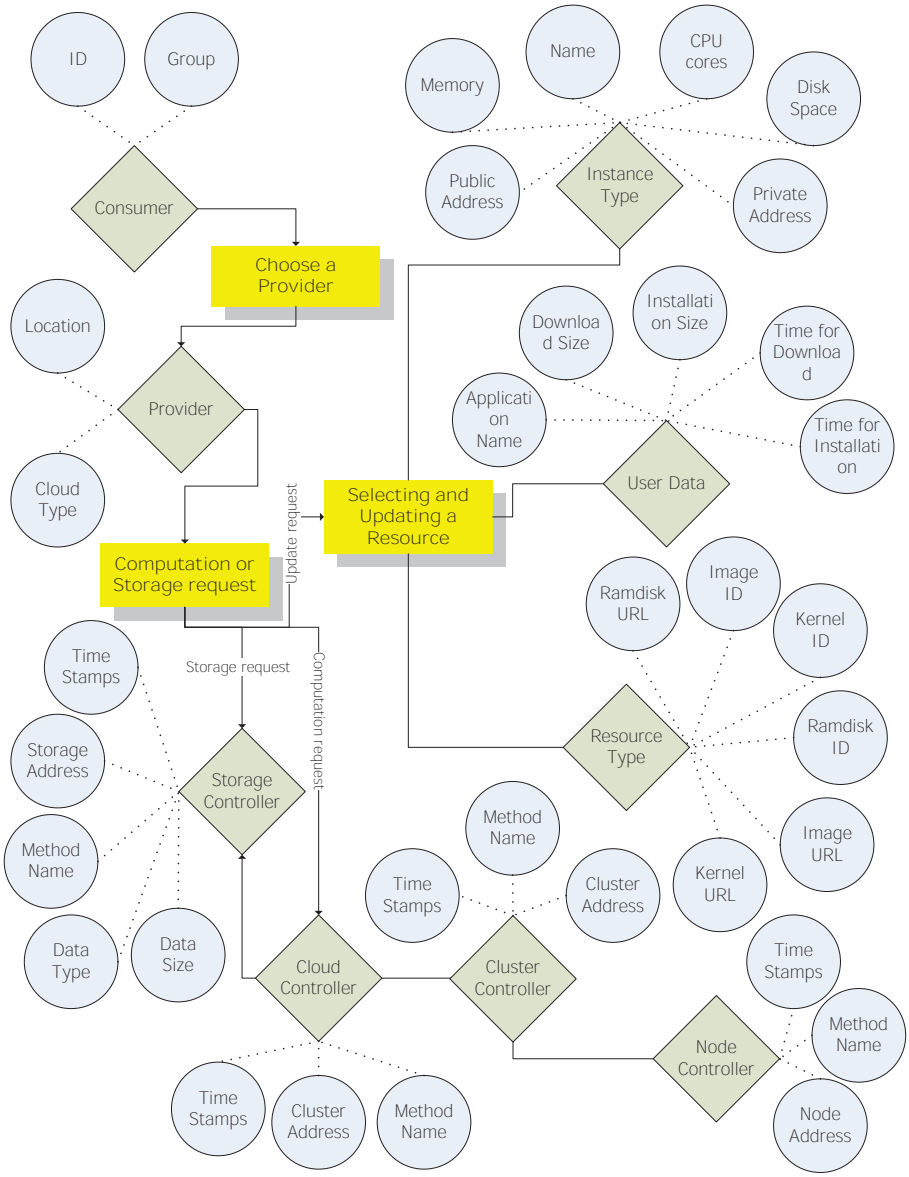


Figure 4.6: Various parameters (provenance data) for the Infrastructure (IaaS) part of a Cloud environment. The **rectangles** represent the decision process and routing a particular request when acquiring a resource. The **diamonds** represents main items (provenance data types) of a Cloud IaaS and the **circles** represent various provenance data from different types of items.

Table 4.2 describes a brief list of various provenance data, their names, values and description which is required in Cloud computing.

Data Type	Data Name	Data Value	Description
System provenance	Image ID, Image URL	emi-39A51609, http://131.130.32.11:8773/services/Walrus/ubuntu-image-bucket/ubuntu.9-04.x86-64.img.manifest.xml	Describes the type of the resource and the physical location of the resource in Cloud.
Cloud process provenance	Reservation ID	r-ABSD987	The ID which is assigned to the resource by Cluster Controller before passing it to a Node Controller.
Cloud process provenance	Instance ID	i-4BC9095C	The ID which is assigned to the running instance from CC and NC services
Cloud process provenance	Service name, Method name	Eucalyptus NC, EucalyptusNCRunInstance	Web service and method name. In this case NC is the service and RunInstance is the method in provenance store.
Instance provenance	Instance type	memory = 192, cores = 1, disk = 2	Describing the instance type with users requirement of memory, processor cores and hard disk space.
Consumers of Cloud	User ID, User group	admin, default	Who is initiating the resource and which group the initiator belongs to.
Cloud user data	User data	Custom script	The information or scripts which are uploaded by users to populate the resources before they are booted and assigned.
Timestamps	Time stamps	Fri Apr 13 15:35:00 2012, Fri Apr 13 15:35:02 2012	Start and end time for a particular service and methods
Other	From, To	131.130.32.12, 131.130.32.11	Sender and receiver IP address for Cloud services.

Table 4.2: Various metadata and their names, values and description for Clouds infrastructure

4.2.4 Provenance Storage

Provenance storage depends on two key points. Firstly, the mechanism to store provenance like copying the original objects or making reference to the original objects in provenance data. Secondly, the technique to store provenance like XML schema or database storage. Below, we provide the details of provenance storage of the framework.

Provenance storage mechanisms

Various mechanisms for storing the provenance data are based on tightly-coupled or loosely-coupled strategies. The tightly-coupled strategy stores the provenance data along with the original data, i.e., provenance and original data are kept together. The advantages of using such a strategy are:

1. Less storage overhead because there is no copy created of the original data in provenance.

2. Easy to manage because provenance data and the original data is placed together.

The disadvantages of such a scheme are:

1. Deleting the original object will result in deletion of the provenance data.

2. Since provenance is stored with original data (tightly coupled) therefore, efficient query and data coupling are challenging.

The loosely-coupled strategy offers the storage of provenance data independent of the original data. Using this strategy, a copy of original data is created in provenance store. Such a scheme can suffer from huge storage overhead when fine grained provenance data is collected. However, the independence of original data and provenance leads to fast and efficient query mechanisms. Contrary to the tightly-coupled strategy, deleting the original objects does not affect the provenance data because copy of the original object is already created. We propose a hybrid approach, i.e., link based mechanism to store various provenance data from the above strategies.

A hybrid approach - link based mechanism

In this scheme, we make a copy of the names of objects and use links or references for original objects in provenance data. Therefore, provenance data is independent of the original data and provides fast and efficient query mechanisms. Moreover, the hybrid approach is not affected from huge storage overhead because only links are

Listing 4.1: Sample of provenance data collected from the Eucalyptus Cloud which describes the structure of stored provenance in XML file

```
<Inflow> !- the name of the flow, i.e., request message-
  <ServiceName>  !- the name of the service-
    <MethodName> RunInstance </MethodName> !-the name of the method-
    <ImageID> emi-39A51609 </ImageID> !-the name of the resource-
    <ImageURL> http://131.130.32.12:8773/services/Walrus/ubuntu-image/
        ubuntu.9.04x86-64.img.manifest.xml
    </ImageURL>
    !-a link reference to the orignal object-
  </ServiceName>
</Inflow>
```

maintained in provenance data. Using the hybrid approach, we define two kinds of relationships that are maintained in provenance data for Cloud computing.

The first relationship is between the provenance data and original data/objects. For instance, when data is stored in Eucalyptus Cloud using Walrus, the creator could be a user and/or Cloud services like CLC. To maintain an overall relationship between provenance and original data along with the creator we create (i) a copy of the creator such as user name and group information, (ii) a copy of the name of the object which is stored, and (iii) a link reference to the location of the object.

A sample *XML* is shown in Listing 4.1, which describes the structure of stored provenance data using the hybrid approach. In the given Listing, *ImageID* is the resource identifier used by Clouds and *ImageURL* is the location of the resource in Cloud storage unit. We make the copy of names such as method name and resource ID and store a reference or link to the original object as presented in the colored tags. The example *XML* is for Eucalyptus[1] Cloud and Walrus service.

The second relationship is within provenance data which is to maintain the flow of Cloud services execution. This relationship is maintained by storing the provenance data in form of a Directed Acyclic Graph (DAG) [130]. Inside DAG, the flow of Cloud services is maintained by linking the child objects to the parent object. For instance, in a Eucalyptus Cloud, when a user makes a request for a resource, the flow of execution is from the user to the resource via Cloud services. Inside the Cloud, the flow is from CLC to the Node service which describes the various layers of Cloud infrastructure. Each layer of the infrastructure adds provenance data and it is collected and stored in a tree structure as depicted in Figure 4.7. The blue circles represent the flow of Cloud services, the green circles represent the main provenance items and the silver circles represent metadata of each item in Figure 4.7.

[1] http://open.eucalyptus.com/

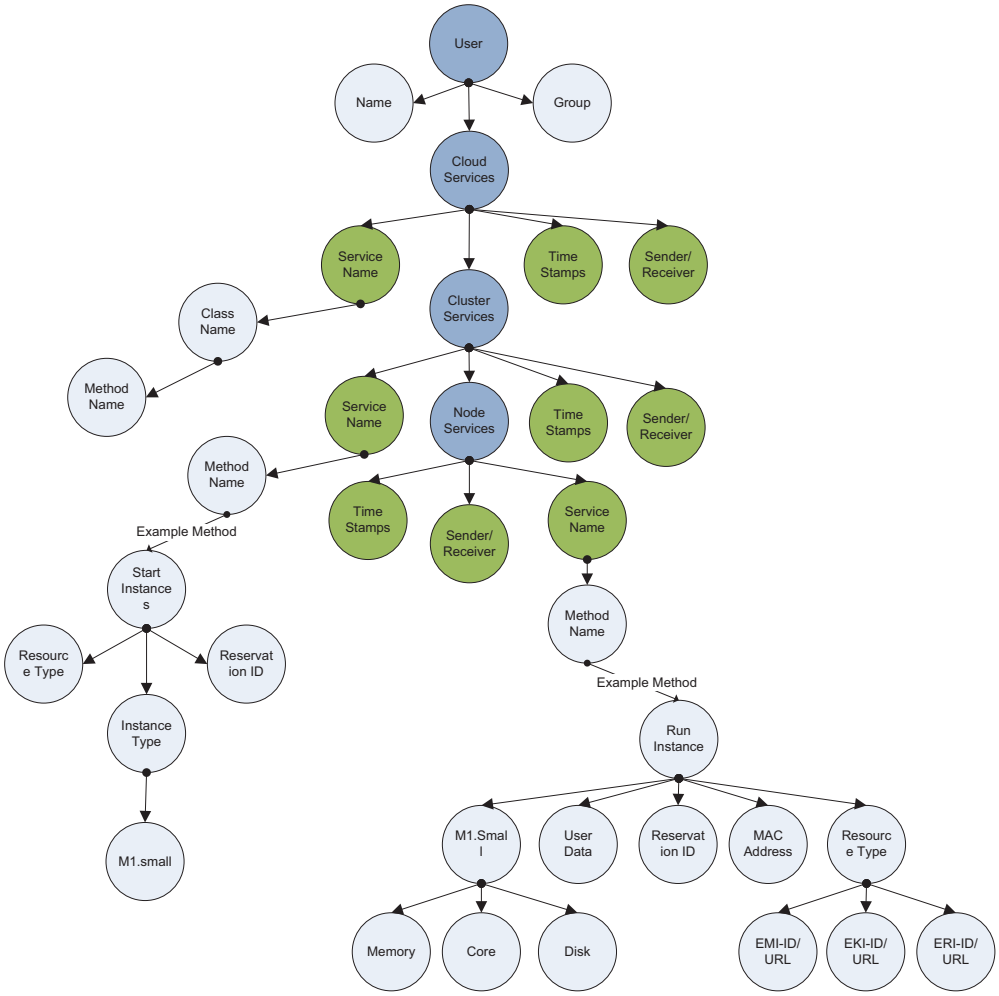


Figure 4.7: DAG mechanism used to store and present Cloud provenance data.

Provenance storage techniques

We propose two methods to store the collected and parsed provenance data. The first method utilizes the *XML* schema to represent and store the provenance data. The second method utilizes the NoSQL database, M/DB [8] in particular for the storage and management of provenance data. The black box architecture to store the provenance data is presented in Figure 4.8. We provide an overview of these storage methods as:

1. **XML storage of provenance:** In this method, we provide a mechanism to store the provenance in *XML* file. We follow an approach where provenance is saved in the same repository or virtual pool that is used to store original objects. For instance, the Walrus service saves users data at a particular

location in Eucalyptus Cloud. We save provenance data at the same location where original data is stored. More precisely we are only utilizing the same location. Our provenance data is still independent of original data.

2. **M/DB storage of provenance:** In this approach, we are using an alternative to the Amazon SimpleDB called M/DB. According to [8], M/DB is not a mock service but a true database and we configured it in the same Eucalyptus Cloud where other services are running. It is deployed on a Virtual Machine instance running Ubuntu Lucid. Just like SimpleDB, M/B provides indexing feature. The difference between the two is the consistency model. SimpleDB supports a weaker kind of consistency called eventual consistency while M/DB is immediate consistent. In this method, provenance is stored in a domain and a SELECT like operation can be performed to query the provenance data.

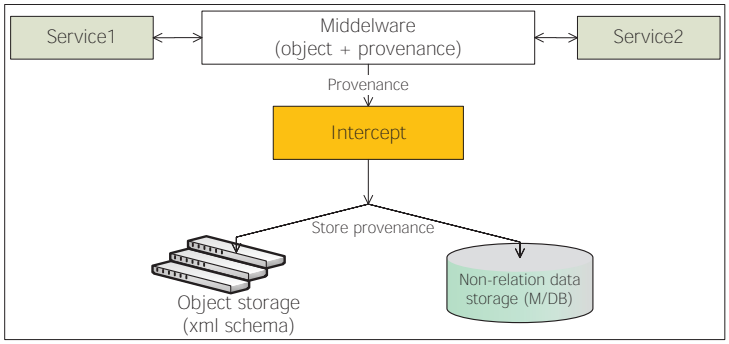


Figure 4.8: Different protocols to store the provenance data

4.2.5 Provenance Query

The query module of the framework utilizes the stored provenance to extract various information. Since provenance is managed independent of the original data, custom applications can be designed to query provenance data based on the individual requirements. For instance, a query can be generated to find the activity pattern in a Cloud IaaS based on the resource types, instance types, time used or user IDs. The results of such a query will generate information which is useful to monitor a Cloud IaaS such as to move the frequently used resources to a faster CPU/disk unit for better performance. Algorithm 1 is an example query to find activity patterns based on the resource-ID.

We devised two different methods to extract the data from the provenance storage which depends on the storage model, i.e., *XML* schema or NoSQL database.

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 ← ChildNode(<ImageID>)
end while
End

In the first method, we used Language-Integrated Query (LINQ) to *XML* to extract information from provenance. LINQ to *XML* provides an in-memory *XML* programming interface that leverages the .NET Language-Integrated Query (LINQ) Framework. LINQ to *XML* utilizes the latest .NET Framework language capabilities and is comparable to an updated, redesigned Document Object Model (DOM) XML programming interface[2]. The second method utilizes Amazon APIs of SimpleDB and provides a SELECT like feature to extract provenance information. Various queries that extract important information from *XML* store or NoSQL database are provided in next chapters.

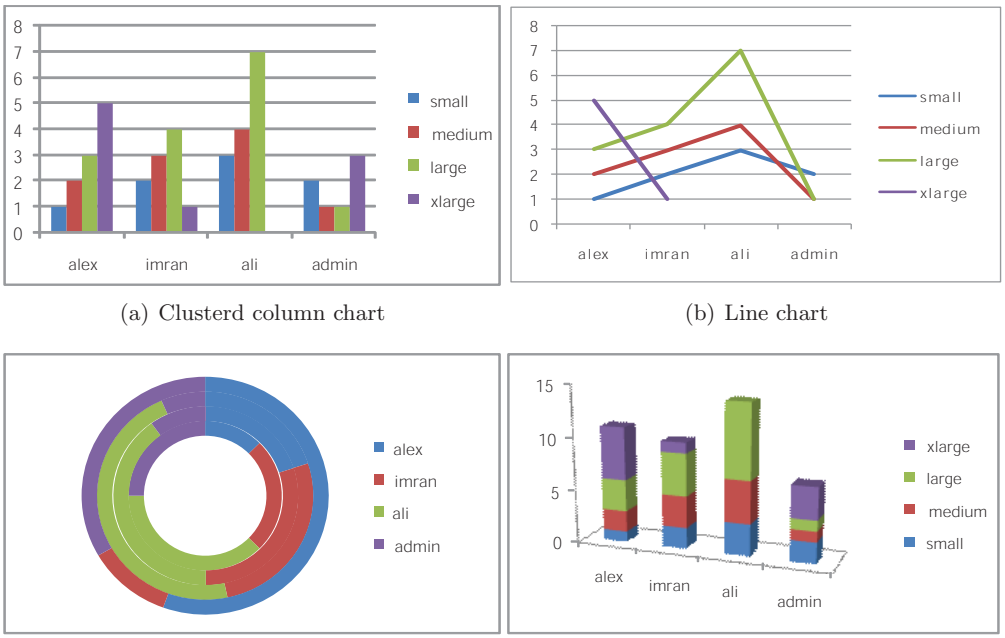
4.2.6 Provenance Visualization

The visualization component takes the query as an input parameter. A particular query is analyzed to find various components and their relationships within provenance data. For instance, a sample query:
Visualize the instance types of various users from the last 48 hours.
is analyzed to find the relationship between the users and various instances requested by them over a specified period of time. The result of the query is visualized in chart form that can be changed at run time with different types of charts, e.g., lines, graphs and pies etc. Figure 4.9 presents a sample visualization of the above query in various shapes.

4.2.7 User Interface

The proposed provenance framework can be engaged to different service models and therefore addresses the respective view points of a user, developer and resource provider. To engage the provenance framework, editing is required for various configuration (XML) files in Clouds. Various audiences can enable/disable the provenance

[2]http://msdn.microsoft.com/en-us/library/bb387098.aspx



(a) Clusterd column chart

(b) Line chart

(c) Doughnut chart

(d) Stacked column chart

Figure 4.9: Visualization of a query in various formats

framework according to their requirements. For example, if we consider the Cloud provider, the list of options is following:

- To enable/disable the provenance module for all clusters
- To enable/disable the provenance module for a particular cluster
- To enable/disable the provenance module for a particular node or all nodes
- To enable/disable the provenance module for selected methods of a particular cluster or a node.

These options make the usability of the proposed framework very high. Furthermore, we developed a Graphical User Interface (GUI) that can be used to engage provenance framework as per various requirements. The GUI facilitates a user in engaging provenance framework and saves the time required to find and edit various *XML* configuration files. Figure 4.10 presents a prototype of the user interface which is available to a Cloud IaaS provider.

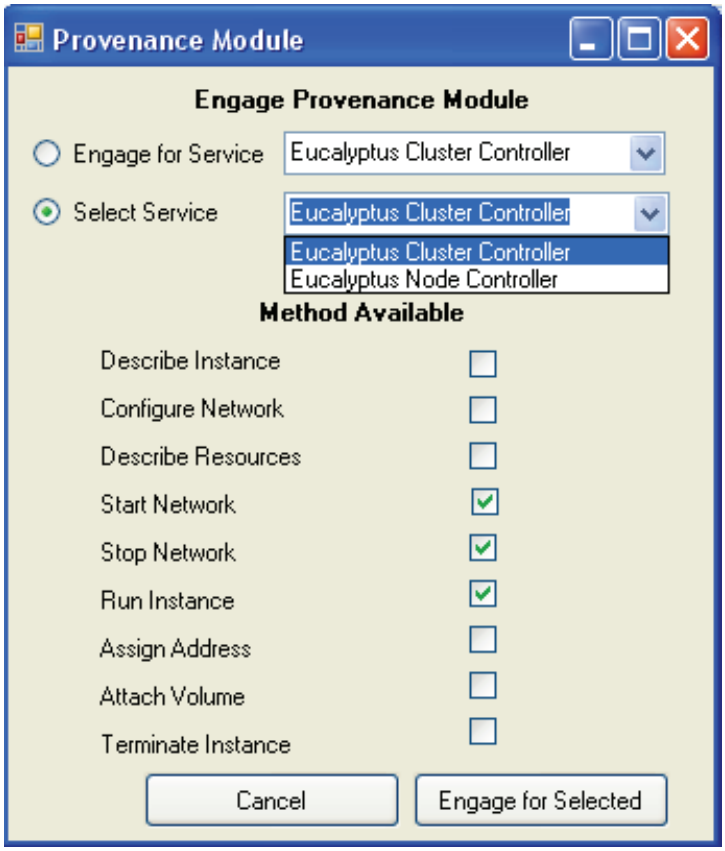


Figure 4.10: User interface to engage the provenance module

4.3 Relationship of the Framework with Provenance Requirements

The proposed approach of interpreting various data on the middleware level and dividing the framework into multiple autonomous modules address various requirements of the framework as shown in Figure 4.11. The following items present some of the relationships in Figure 4.11.

- Intercepting the provenance data on middleware layer address abstraction property of Cloud because data collection is not part of the platform, infrastructure and software services. Moreover, the interception at middlewares makes the approach independent of Service Oriented Architectures.

- Modularity of the proposed framework makes our approach available to different services models such as IaaS, PaaS and SaaS because with each model, only the collection part needs to be modified and rest of the components remain

the same.

- The consistency of the framework helps us to integrate provenance data from different layers of Cloud and hence explore the relationships among various layers.

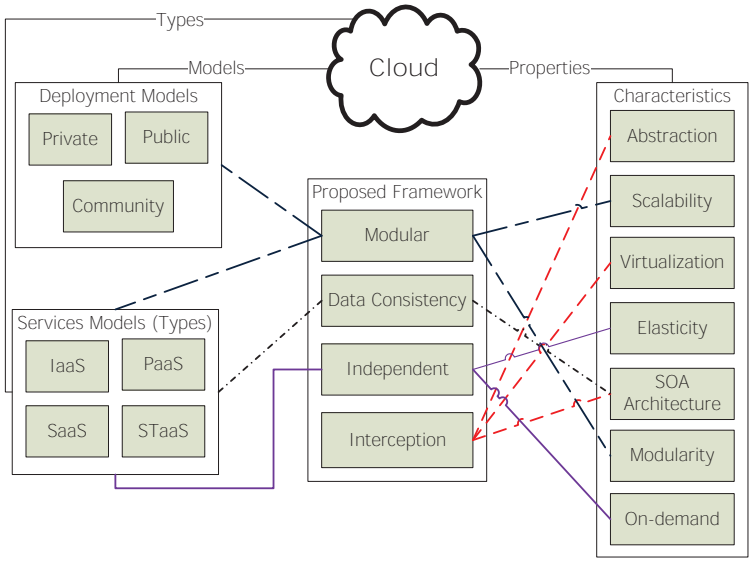


Figure 4.11: Various relationships between the components of the framework and requirements of provenance in Cloud.

4.4 Provenance Enabled Clouds: Implication of the Framework

Any framework is useless without presenting the implication in its own context. Therefore, to present applications which utilize various components of the framework is extremely important. Provenance enabled Clouds have many advantages depending on the service models. For instance, introducing the provenance data into a Cloud IaaS has the following advantages:

- **Efficient searching:** Using the provenance data of Clouds storage such as Eucalyptus Walrus to search and find contents. For instance, collecting metadata of Walrus and utilizing the metadata to provide fast, efficient and user controlled search mechanisms.

- **Reports generation:** Utilizing the provenance data of Clouds to generate various usage reports. For instance, generating reports about the usage of re-

source types, instance types, memory, storage, user data, volumes, and snapshots for a Cloud IaaS. These reports can be based on a specified time period.

- **Patterns:** The use of provenance data to find similarity patterns that exist in instance types, resource types and/or users data in a Cloud. Provisioning techniques can utilize such patterns to forecast future requests from various users. Similarly, the collected provenance can be analyzed to find the behavior of various users in storing, accessing or managing contents in Clouds, e.g., finding famous or most used contents.

- **Resource utilization:** Utilizing the provenance data for efficient planning and execution of resources in Clouds. For instance, when a user makes a request for a resource with particular requirements such as resource type, instance type, and user data, the provenance data is queried for a match, i.e., if such a resource already exists, a copy of the running resource is allocated to the user instead of creating a new resource from scratch.

- **Reduced cost and energy consumption:** Provenance enabled Clouds result in reduced cost and energy efficiency using factors such as similarity patterns to forecast future requests, utilizing existing resources and using provisioning techniques for efficient planning of resources.

The introduction of provenance data into platform and software layers has many advantages such as:

- **Trust, reliability and data quality:** Trust, reliability and data quality in distributed computing are concerns of any organization and end users. For instance, how to identify if the data produced by users and their applications is trustworthy. Provenance enabled Clouds provide the opportunity to test and verify the final results based on the source data and the applied transformations along with the information about users. Therefore, trust and reliability of provenance enabled Clouds is higher when compared to a Cloud that is not provenance enabled.

- **Fault detection:** Utilizing the provenance framework to pinpoint the exact time, service, method and related data in case of a failure. This can be achieved for any type of service model, i.e., IaaS, PaaS and SaaS.

- **Verification:** Utilizing the provenance data to verify the final results of any application by detailing and verifying the individual steps of applications.

- **Layering of provenance:** Provenance can be combined from various layers of Clouds for better understanding of the underlying architectures such as exposing the relationships between layers. Therefore, integrating (layering) the provenance data from various layers is another important and significant aspect of Cloud computing.

It is important to note that the above advantages are just descriptions of many more that can be achieved by incorporating provenance into various service models. Various applications utilizing the developed framework and its various components are detailed in Chapter 5.

4.5 Provenance Framework: Implementation of the Interceptor Approach in Cloud IaaS

To understand the proposed framework, firstly we provide a brief overview to the most important Mule and Axis2/C architectures that work as middlewares in Eucalyptus Cloud. Secondly, we provide the details of the interception mechanism for the collection of provenance data according to different middlewares. Afterwards, we detail our framework which consists of components, i.e., *provenance collection*, *provenance parsing*, *provenance storage* and *provenance query*. A few steps logic presented below is used to configure the interception technique along with other components of the framework to the Eucalyptus Cloud.

Extension of the middleware of different service models (IaaS, PaaS, SaaS, STaaS) in Clouds

Require: Various middlewares on different layers (e.g., IaaS layer) of Cloud
Ensure: Middleware is working properly
Begin
Step 1 Check out the middleware
Step 2 Create a module named provenance
Step 3 Modify the module with custom logic such as provenance collection
Step 4 Embed further logic such as provenance parsing and storing
Step 5 Update the middleware by embedding the new module
Step 6 Configure various configuration files with the newly added logic
Step 7 Repeat Step 1 to Step 6 for other middlewares on the same layer
Step 8 Repeat Step 1 to Step 7 for middlewares on all the layers of a Cloud
End

Using the above steps, we configure provenance for the Eucalyptus Cloud. The CLC services of the Eucalyptus Cloud are deployed using a Mule framework. These 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 the Mule framework messaging protocols (HTTP, SOAP, XML, etc.) to communicate with each other and with other Eucalyptus services such as NC and CC. The NC and CC services are deployed using the Axis2/C framework. The subsections below present the architecture of Mule, Axis2/C, their extension through interception mechanism and the configuration of provenance to various components of the Mule and Axis2/C.

4.5.1 Mule Enterprise Service Bus

Mule is a lightweight Enterprise Service Bus (ESB) written in JAVA and is based on the Service Oriented Architecture (SOA) [53]. Mule enables the integration of different applications regardless of their communication protocols. To understand the Mule architecture, we present the main concepts that are called *building blocks* which are related to the proposed framework.

- **Flow:** Mule ESB accepts and processes messages from various applications where the messages are transformed step by step using a series of individual components. The orchestration of these individual components constitutes a flow. Any flow in mule supports synchronous, asynchronous, request-response and other types of communication. The facility to create child-flows inside a flow is also possible. The integration of various flows together builds the actual application. When a message enters to a particular flow, it passes through various individual components, e.g., an input message in *XML* format is transformed to new contents of *text* format and the response is generated to the source (client).

- **Mule Message:** The actual data that is flowing through various components of an application in different flows is called a Mule Message [54, 55]. This message consists of two parts which are called *header* and *payload*. The message *header* contains important metadata such as properties and variables that are used to get the message to the appropriate target. The properties and variables share the same format of name-value pair where name is the key and value are the contents in a message header. The message *payload* contains the actual content or data which is being transported and flowing via applications. When a payload goes through different flows, additional metadata is added, e.g., security parameters thus enriching the existing payload by adding more data.

- **Routers:** Routers are used to control the processing of incoming and outgoing messages by different components in Mule [56]. Two of the important routers

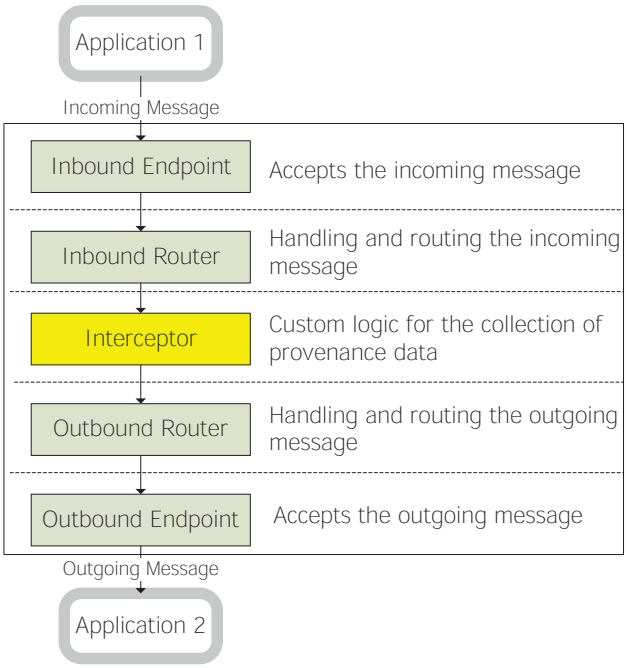


Figure 4.12: The architecture of interceptor in the Mule framework

in the context of this thesis are called *Inbound* and *Outbound* routers. The *Inbound* router is used to handle the incoming message and the *Outbound* router is used to handle the outgoing message by different services and components.

- **Mule Interceptor:** An interceptor is a piece of code in Mule ESB which can be attached to various components or building blocks to alter the behavior of the messages [57]. The interceptor allows intercepting the processing of an object at run time. The basic uses of interceptors are permission checks and security checks for various components. The interceptors can be dynamically configured in Mule ESB for different applications and their components (flows). Figure 4.12 presents the basic architecture of intercepting the data (Mule Message) when two applications communicate via Mule framework. The Mule message could be for any component, e.g., a web service and/or any event.

Extending the Mule Framework (for Eucalyptus Cloud)

The Mule framework is based on a layered architecture and modular design. Mule offers different kind of interceptors such as *EnvelopeInterceptor, TimeInterceptor* and *Interceptor* to take the control and edit the incoming or outgoing message inside a flow. Provenance is metadata information which is flowing between different

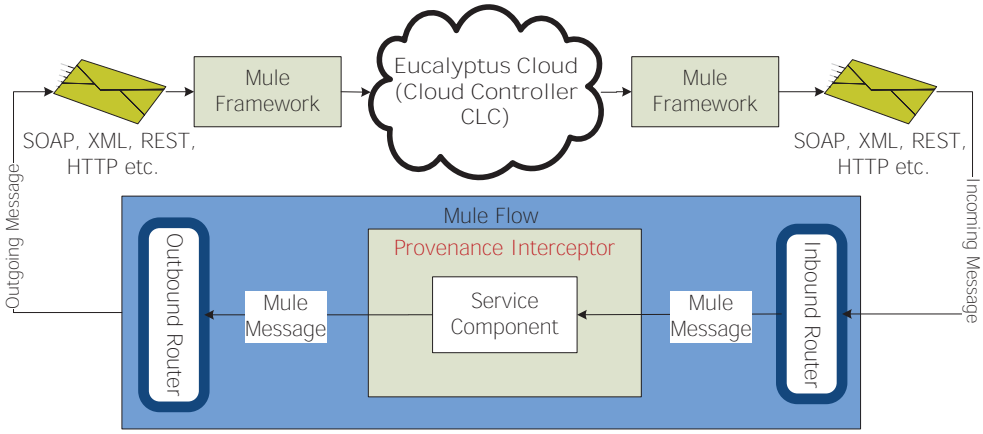


Figure 4.13: Mule Interceptor for a particular component inside a Flow

components and services and we do not need to edit the message structure therefore, we use the *EnvelopeInterceptor*. The Envelop interceptor carries the Mule Message and it is executed before and after a service is invoked. Figure 4.13 describes the architecture of Eucalyptus CLC services where the Mule framework is used as a middleware to communicate between different services. As shown in Figure 4.13 the provenance interceptor is placed between the incoming and outgoing routers inside a particular flow of a Mule.

Configuring the Mule Interceptor (for Eucalyptus Cloud)

There are two steps involved in configuring Mule interceptors to Cloud (IaaS) services. The first step is to compile and build a provenance package (JAVA class files) and copying them to the services directory of a Cloud. The second step requires editing the 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 4.2 is a sample "*eucalyptus-userdata.xml*" Mule configuration file which is used to verify users credentials and groups. In the Listing 4.2, we provide the mechanism to add provenance into a Mule configuration file.

There are various other components in CLC services using the Mule framework for communication and linking between them and with other IaaS services. Appendix A.1 provides a list of various configuration files used by Eucalyptus CLC services.

4.5.2 Axis2/C Architecture

Apache Axis2/C is a C language implementation for the SOAP model of communication which is platform independent and portable. Axis2/C also supports the REST

Listing 4.2: Configuration of the provenance module in the Mule framework

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org/...">
  <interceptor-stack name="CLCProvenance">
!-indicating the interceptor stack created by us for the collection of
    provenance data-
    <custom-interceptor class="eucalyptus.CLCprovenance"/>
!-indicating the path of the java package and class name for CLC
    services provenance data collection-
  </interceptor-stack>
  <model name="eucalyptus-userdata">
    <service name="KeyPair">
      <inbound>
        <inbound-endpoint ref="KeyPairWS"/>
      </inbound>
      <component>
        <interceptor-stack ref="CLCProvenance"/>
!-configuration of the "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>
```

model of web services. The communication between various web services that are deployed by using Axis2/C is achieved via XML data format for the input and output parameters. In the context of this thesis, we describe the following concepts of Axis2/C and the mechanism they are layered and linked with each other:

- **Flow:** In Axis2/C, a flow is a logical concept of the execution of messages. It is a collection of phases (explained next) that work as a real execution chain. There are two main flows which are named *Inflow* and *Outflow* in Axis2/C. As the names suggest, Inflow hold the incoming messages and Outflow hold the outgoing messages. There are two other flows which are named *Infaultflow* and *Outfaultflow* and they deal with the transactions which are erroneous or faulty for the incoming and outgoing messages.

- **Phase:** A phase is a collection of handlers (explained next). The orchestration of different phases constitutes a Flow and they are invoked in order of placement. Invoking a phase would result in invoking of all the handlers that are inside a particular phase. The execution of handlers is also based on the order of placement inside a phase. There are predefined phases in Axis2/C which are

Listing 4.3: The orchestration of built-in and user-defiened phases in Axis2/C

```
<?xml version="1.0" encoding="UTF-8"?>
!- The incoming message is intercepted here-
  <phaseOrder type="inflow">
!- predefined phases in Axis2/C-
    <phase name="Transport"/>
    <phase name="PreDispatch"/>
    <phase name="Dispatch"/>
    <phase name="PostDispatch"/>
!- user defined phases where custom interception is implemented for
    incoming messages-
    <phase name="userPhase1"/>
    <phase name="userPhase2"/>
  </phaseOrder>
  !- The outgoing message is intercepted here-
  <phaseOrder type="outflow">
!- user defined phases where custome interception is implemented for
    outgoing messages-
    <phase name="userPhase3"/>
    <phase name="MessageOut"/>
    <phase name="userPhase4"/>
  </phaseOrder>
```

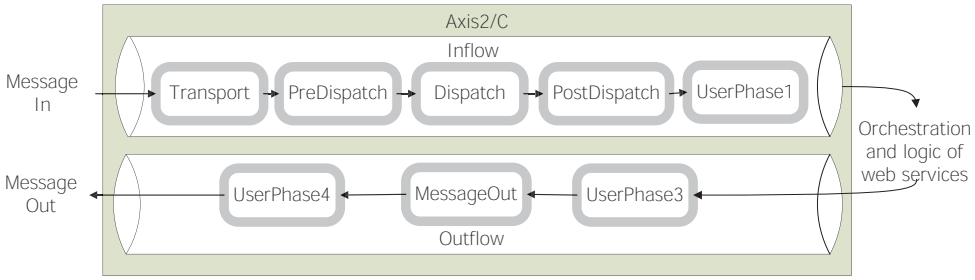


Figure 4.14: Architectural overview of Phases and Flow in Axis2/C

named *Transport*, *Pre-Dispatch*, *Dispatch* and *Post Dispatch* for the Inflow. In Outflow the corresponding phase is named *Message Out*. In addition to the pre-defined phases, a user can add a custom (user-defined) phase in Axis2/C. Listing 4.3 presents an example of the configuration file of Axis2/C and the ordering of phases inside a flow. The phaseOrder element's type identifies the flow name in Listing 4.3. Figure 4.14 presents the architectural overview of Axis2/C, Inflow and Outflow channels and the corresponding pre-defined and user-defined phases.

- **Handler or Interceptor:** A handler or interceptor is the smallest execution unit in the Axis2/C engine. The invocation of a phase results in the execution of each handler in the chain of handlers. A handler can perform any kind of processing, e.g., quality of a service aspect of web services. One of

the important handlers which is used in Axis2/C is for the WS-Addressing [5] implementation. This handler is divided into two parts which are named *AddressingInHandler* and *AddressingOutHandler* that are placed in the corresponding Inflow and Outflow sections. *AddressingInHandler* is placed in the pre-dispatch phase inside the Inflow where *AddressingOutHandler* is placed in the MessageOut phase inside the Outflow. The addressing hander deals with the processing of incoming and outgoing SOAP headers in the request and response messages which are referred as *Message In* and *Message Out*. Similarly, the WS-Security [4] handler deals with the processing of the incoming and outgoing SOAP body for the encryption and decryption of the messages before presenting them to the rest of the components involved in Axis2/C. A custom handler can be written and placed inside user-defined phases that can perform any kind of processing required by users and applications such as collecting provenance.

- **Module:** The collection of handlers and their configuration is called a module. A module is basically a configuration concept that enables a user to extend the Axis2/C by adding custom handlers. Each module has a configuration file which is named *module.xml* that defines the placement of handlers into various phases and flows. To add a particular module in Axis2/C, it is required to write a program (the logic for a particular purpose) in C language and implement the various interfaces. To consume a module, the configuration file is required, and Listing 4.4 provides the sample *module.xml* configuration file for Addressing handler. In Listing 4.4, *class* is the C (programming language) implementation and *module name* is the given name to the module.

  Once the required classes are written in C language and the configuration file is ready, the next step is to deploy the module in Axis2/C. The deployment step requires creating a folder with the same name as the module name and putting them together in the appropriate repository of Axis2/C. The final step in utilizing a module is to engage the module in Axis2/C. A module can be engaged as following:

  1. Globally to all the web services of Axis2/C by configuring the *axis2.xml* file.
  2. To a particular web service in Axis2/C by modifying the *services.xml* file of a web service.
  3. To a particular method of a web service.

  Figure 4.15 presents the extension of Figure 4.14 with the implementation of

Listing 4.4: A sample configuration file required to configure Addressing Handler

```
<?xml version="1.0" encoding="UTF-8"?>
<module name="addressing" class="axis2_mod_addr">
    <inflow>
!- configuration of a handler to inflow phase with handler details like
    name and class-
        <handler name="AddressingInHandler" class="axis2_mod_addr">
            <order phase="PreDispatch"/>
        </handler>
    </inflow>
    <outflow>
!- configuration of a handler to outflow phase with handler details
    like name and class-
        <handler name="AddressingOutHandler" class="axis2_mod_addr">
            <order phase="MessageOut"/>
        </handler>
    </outflow>
</module>
```

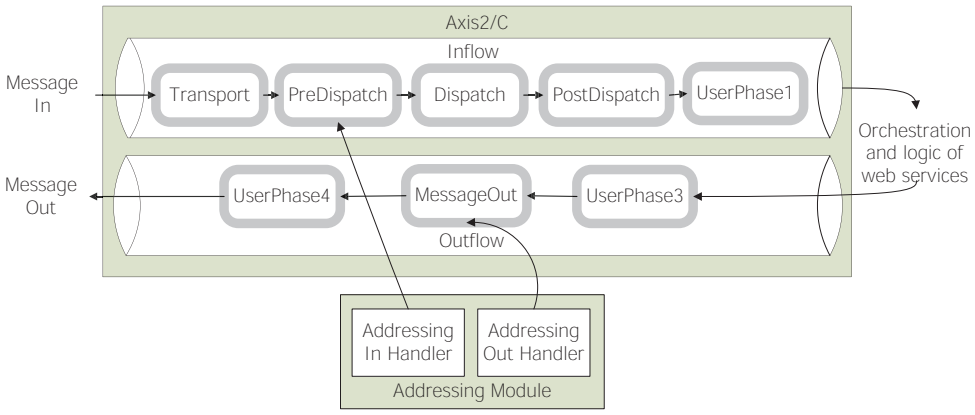


Figure 4.15: Configuration of a Module in Axis2/C

Addressing module into appropriate phases and flows of Axis2/C.

Eucalyptus Services and Axis2/C

The Node and Cluster Controller services in Eucalyptus IaaS are exposed to other components and services by using Apache Axis2/C framework. We intercept the data using handlers and modules [62] from the underlying linking and communication framework (Axis2/C) and thus extend the Cloud in a seamless fashion. The messages flow between components of CC, NC and CLC where they are linked through Axis2/C engine. Our custom handlers are deployed for the collection of significant provenance for each service and corresponding activity of the service.

The interception technique has been used already for many purposes such as web service security and addressing [4, 5] in Apache Axis. Similarly, the concept is

Listing 4.5: Configuration of provenance into Axis2/C

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

used in [71] for SOAP services with WSDL configuration in workflow execution where services are deployed in a Tomcat container for the collection of provenance. As with other related work from grid and workflow for the collection of provenance data, this work is focused on the application layer, i.e., workflow execution. However, we are interested and motivated by the provenance of infrastructure, platform, software and data, i.e., provenance of Clouds. Therefore, the work in [71] is not extensible to Cloud services and platforms because of the different domain, communication mechanism, and underlying architectures, i.e., Cloud, REST protocol, and Axis2/C and Mule.

Configuration:

The proposed module for the collection of provenance data can be configured globally to NC and CC services by editing the *axis2.xml* file, or to a particular service and method by modifying *services.xml* file. Listing 4.5 describes the configuration of provenance module to Eucalyptus NC service.

The complete configuration files of the services that utilize Axis2/C in Eucalyptus are provided in Appendix A.2. Appendix A.2.1 and A.2.2 provides configuration files for Eucalyptus Cluster and Node Controller respectively. Similarly, Appendix A.2.3 provides the complete *axis2.xml* file to configure the provenance module globally in the Eucalyptus Cloud.

4.5.3 Provenance Configuration

The modules for provenance collection are placed inside the Apache Axis, Axis2/C and the Mule frameworks (middlewares) of Cloud service modules such as Euca-

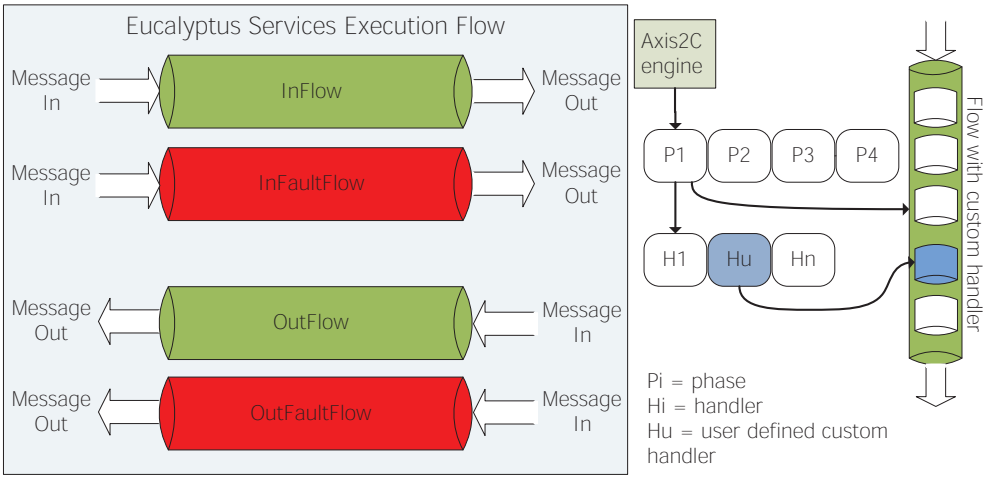


Figure 4.16: Framework components for CC and NC services.

lyptus and WSO2. When a message enters in Eucalyptus Cloud, it goes through CLC, CC and NC services respectively. The provenance module and corresponding handlers are invoked depending on the particular service, e.g., Mule handlers for CLC services and Axis2/C handler for CC and NC services. For example, when we consider the Axis2/C engine and a message enters to *Inflow*, all the handlers are invoked inside the Inflow. *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 the Eucalyptus Cloud (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 Eucalyptus Cloud Axis2/C engine and the execution of a service with input and output messages is described in Figure 4.16. The left side of Figure 4.16 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 Eucalyptus Cloud IaaS execution chains. When a component inside Cloud IaaS is invoked, provenance collection module intercepts the flow and gets the hold of the message for provenance data in the corresponding execution flow. Similarly, the flows inside Mule framework are intercepted by using the Mule *EnvelopInterceptors*. The *EnvelopInterceptor* hold the message which is passing between the components of CLC and also when the message is passed from CLC to CC service in Eucalyptus Cloud.

Various components of the framework such as *provenance storage*, *provenance*

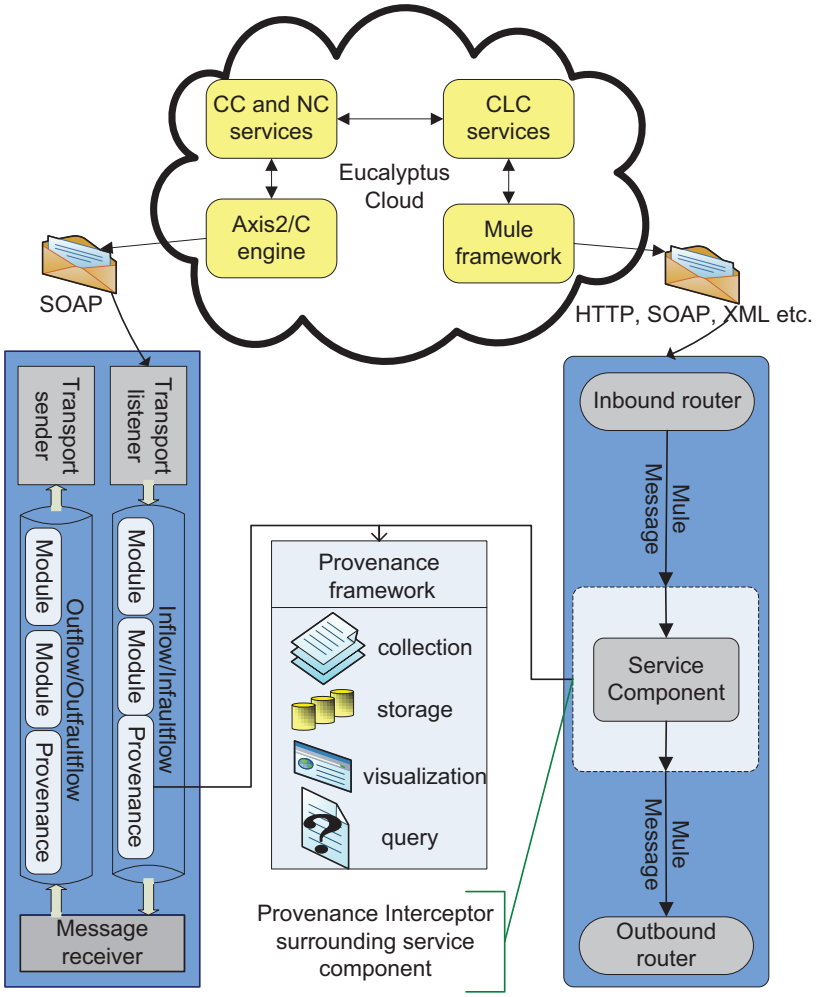


Figure 4.17: Framework components

query, and *provenance visualization* in the context of the Eucalyptus Cloud IaaS are presented in Figure 4.17. Figure 4.17 further details the interaction mechanisms between Cloud services which are using different communication protocols and frameworks such as Apache Axis2/C and Mule.

4.6 Framework Experience

The extension of middleware, i.e., Apache, Axis2/C and Mule by exploiting the built-in features of handlers and modules facilitated in the provenance collection that is independent of Clouds, their architectures, and types such as IaaS, PaaS and SaaS etc. We followed a modular approach and divided our framework into various independent components to address different requirements and the defined standards.

We believe that the future of provenance in Clouds lies in a lightweight and independent provenance scheme to address cross platforms, different types of Clouds, and application domains. Furthermore, the proposed framework can be deployed without making any changes to the Cloud services and underlying architectures. Following are the advantages of our approach and the proposed framework:

- It is independent of Clouds such as various service models (IaaS, PaaS, SaaS). The framework is compatible with any domain where Apache, Mule or similar frameworks are utilized as middleware.
- The proposed framework follows a soft deployment approach and therefore, no installation is required for any component of the framework.
- Various challenges and requirements are offered by Clouds such as *virtualization*, *on-demand computing*, *pay-as-you-go model*, *abstraction*, *extremely flexible* architecture and the inability to extend Cloud services and architectures. The proposed framework addresses these requirements in an automatic fashion because it becomes part of the Cloud middleware. For example, adding a Node or Cluster to the existing infrastructure (Eucalyptus) simply require to engage the provenance module for the newly added Node/Cluster.
- The proposed framework is extensible to Platform and Software layers by adopting the same guidelines as described for IaaS in this chapter.
- The proposed framework requires minimal knowledge and understanding of the underlying service models and architectures of Clouds. Furthermore, trust is assured by augmenting a Cloud infrastructure with provenance collection in a structured way.

Major disadvantage of the proposed framework is:

- The proposed framework relies completely on the extension of the tools and libraries, i.e., the middleware and cannot work in any service model where the middleware is not extensible.

4.7 Conclusion

With the evolution of distributed computing and the shift of compute or data science towards Clouds, it is challenging to keep the important information about applications, platforms, infrastructure and data in Cloud computing. In this chapter, we described a list of requirements and standards that must be addressed when collecting provenance data in Clouds. Based on the list of requirements, we proposed

a provenance framework with properties such as modular design and seamless approach for the collection of significant provenance data at various layers of Cloud computing. Hereby, the design of the framework is detailed along with various components such as *collection*, *storage*, *query* and *visualization* which address the individual significance of provenance data. The implementation of the framework module, i.e., *provenance collection* follows which details the underlying architecture and the mechanism to intercept provenance data. This chapter also provide the implication of the framework, i.e., and overview of various application that can benefit from the developed framework.

The characteristics of the framework such as modularity, Independence, and consistent approach enable the integration of provenance in Clouds with minimal knowledge and understanding of the underlying architecture and services. Furthermore, the developed framework can be applied uniformly across all the layers of Cloud computing. Provenance proved its significance in distributed computing such as grids and workflows for verification and audit trials. With the recent trend of technology shift towards Clouds, provenance must be properly addressed in the context of Cloud computing such as provenance of Clouds themselves. Our framework provides the necessary details of how provenance is embed in Clouds without altering the architectures and services.

5. Applications of the Provenance Framework in Clouds

The developed framework collects, parses and stores provenance of Clouds in a seamless fashion using various independent modules. The utilization of the collected provenance and various components of the framework such as adding new capabilities to the existing Clouds is significant from consumers, resource providers and application developers perspectives. In this chapter, we explore various applications where the developed framework and the collected provenance are utilized in the context of different service models of the Cloud.

- Contents Search in Clouds: A user controlled search mechanism for finding various contents in Clouds object storage using metadata (which is a subset of provenance data). The object storage is used for storing persistent content and the developed framework plays a key role in collecting, parsing and management of metadata of various contents. The collected information of the content is further utilized for providing a fast and efficient search mechanism.

- Usage Reports and Similarity Patterns: Utilizing the stored provenance of users, groups and Clouds themselves to generate various usage reports of compute and storage tasks such as memory, object storage and volumes among others. These reports present trends or similarity patterns over time in the usage of Cloud resources from various users.

- Utilization of Resources: The utilization of resources by using the similarities in the resource usage, user activities and other important parameters, e.g., volumes and snapshots. For instance, querying provenance data for future compute and storage requests and finding the similar resources in the existing Cloud, i.e, reusing existing resources (instances to be specific).

5.1 Providing Content Search for Clouds Object Storage Using Metadata

Goal: *To provide a new capability of searching content which delivers an efficient, fast and user controlled search mechanism for Clouds object storage using the provenance framework.*

5.1.1 Introduction

A huge amount of data produced by sensors, mobile devices, social media and other gadgets connected through internet is stored in Clouds [29, 19, 49]. Moreover, the shift of storage of data such as text files, images, and audio and video from personal computers toward Clouds is in progress. Amazon S3 [2], Microsoft Live SkyDrive [9], DropBox [58] and Eucalyptus Walrus [7] are few examples from business and research domains used by customers to archive their personal data in Clouds. These services are used by enterprises, scientists and home users for various purposes due to which the stored data is in different size and various formats. The possible preferences from a consumer could be to store a large audio and video file or small files (e.g., text files) in abundance. The variety and range of the data stored in Clouds is vast and it became challenging to effectively and efficiently find relevant data according to the requirements of users.

Metadata is described as information about information and it could be as simple as an author's name of a document. Among others, metadata is used for searching, content and privacy protection and resource discovery in data or computational science [107, 114, 86]. For instance, the metadata about Cloud services and the performance of those services (submitted by users) from various providers can be used to make better decisions when deploying a particular application in Clouds as proposed in [120].

Metadata is used as one of the key features to search and find meaningful data [83]. In Clouds, the persistent data from various users is stored as objects inside buckets with various Access Control Policies (ACPs) using the storage service such as Eucalyptus Cloud and Walrus service. While storing data, metadata such as file name, size, type, group and user's detail is also associated with the stored data. The amount and type of metadata can vary according to various layers of distributed architectures [45]. For example, the physical layer in a sensor network can provide the information of particular devices such as heat or touch sensors and their models etc. This information can be further extended to other layers such as the operational layer where particular algorithms are used to filter the original

data, e.g., filtering and sampling rules. In the context of this thesis, we focus on the utilization of three types of metadata as following:

1. **System metadata:** The information which is associated with files on the client side, e.g., file name, file format, size etc.

2. **User-defined metadata:** The information which is provided by various users, e.g., notes.

3. **Server metadata:** The information which is associated with the objects on the server side, e.g., owner details.

5.1.2 Problem Description: What is Missing?

Clouds manage the data which is mostly persistent and unstructured via a service called *object storage*. With the huge amount of unstructured data comes the issue of retrieval of data relevant to the requirements of a user. As the volume of personal and commercial data is moved to private or public Clouds, retrieval of meaningful data, i.e., the exact data users require in an effective and efficient manner has become challenging. In current storage offerings such as Amazon, Eucalyptus and Google, the content or objects are defined by a *key-value* pair.

In a *key-value* based storage, there is no native method to search content such as a *Select * From ..Where* operation in traditional databases. However, objects can be listed based on the unique key[1,2], but searching and finding content with any other parameters (metadata) such as type, size, owner, time stamps etc., is not directly supported. Therefore, for searching content, users must know the key or bucket name where the objects are stored. An indirect approach, e.g., *hit and trial* can be taken into consideration which utilizes the existing metadata in Clouds[3]. For instance, metadata can be retrieved through the name (key) of the object in existing Clouds. To search for content, a user needs to loop through all the objects and their metadata until the match is found. This method is cumbersome, tedious and inefficient for large number of objects stored in Clouds. To solve the problem of finding contents in Cloud, we propose to utilize various components of the provenance framework. Hereby, we provide a fast, efficient, easy to use, detailed and user controlled search mechanism.

[1] http://docs.aws.amazon.com/AmazonS3/latest/dev/ListingKeysUsingAPIs.html
[2] http://docs.aws.amazon.com/AmazonS3/latest/API/RESTBucketGET.html
[3] http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingMetadata.html

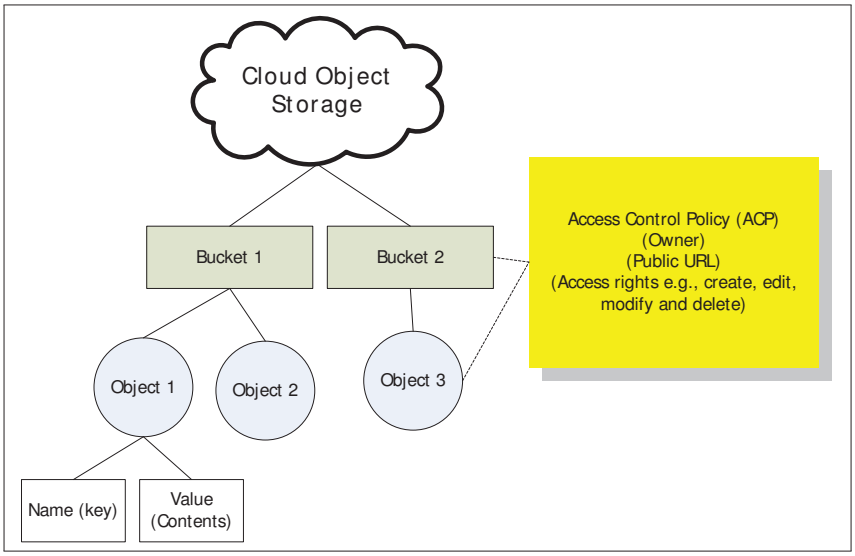


Figure 5.1: The storage architecture of Eucalyptus Walrus

5.1.3 Understanding the Architecture

In Cloud environments, an independent service or model referred to as *object storage* is used for the management of persistent data. This model provides a virtualized pool of storage resources and can be accessed via Application Programming Interface (API) or a web browser. *Object storage* is used to store the persistent data from various users through *put/get* methods of the REST protocol. *Object storage* also stores the virtual machine images in Clouds. Cloud storage services are used by enterprises, home users and researchers to store their files of various size and formats.

The data in Clouds is stored utilizing a hierarchal structure of buckets and objects as shown in Figure 5.1 for Eucalyptus Walrus. A bucket is a container like a folder in file systems and the actual data is represented as objects. While storing objects, some metadata is also stored such as Access Control Policy (ACP) that contains information about the access rights, e.g., creation, deletion, modification etc. as depicted in Figure 5.1. Similarly, each object has a unique identifier, i.e., a key and value attribute and it is also stored as the metadata information.

To search for content in Clouds, i.e., the *hit and trial* approach utilizes a method where the metadata of individual objects is retrieved. The result, i.e., retrieved metadata can be compared with the request parameters from users. To find the exact content, a loop is required which goes through all the objects until the match is found.

5.1.4 Proposed Solution

In this thesis, we are mainly interested in navigational queries, i.e., a user wants to find some important data which is already stored in a Cloud. Our goal is to utilize the description of contents (data) for finding the original or related contents. Table 5.1 presents the various metadata and their description that are utilized for finding contents in Cloud object storage.

Name	Description
Type	The type of different contents such as text files, pdf documents, and audio or video files
Source	The client application or software that is used to upload the contents, e.g., JetS3t
Policy	The ACP (content policy) that is used to share the contents with other users, e.g., public access to read any content
Region	The storage location of contents, e.g., in S3 a location can be set as per users choice to EU and US etc.
User	Information about the person and group details who is responsible to create, update, share or delete any contents
Path	The location of contents, i.e., objects inside the directory structure of Walrus and the unique identifier
Time stamps	The creation, update, deletion and access times of contents in Clouds
Access history	How many times and when was a particular object (content) accessed. This information can be used to move popular contents to a fast storage unit for better efficiency and faster access
Size or Length	This information is used to describe the size of contents for audio and video files, and length for text files, e.g., number of pages
Access URL	This presents the URL of contents when it is shared for access by the owner to other potential users
Custom parameters	User defined metadata that is represented as name-value pair and contains any information such as description of documents

Table 5.1: Various metadata and their description for object based storage

The high level architecture of the solution for providing effective and efficient search mechanism is presented in Figure 5.2. This architecture answers two key questions as following:

- **What is the role of the framework in content searching?** As shown in Figure 5.2, users of the Cloud interact with the object storage from anywhere using private network or internet. The *Object Storage* service such as Walrus manages the contents according to the different operations (CRUD) performed by users. The developed framework seamlessly intercepts the metadata and stores it either in object storage (XML Schema) or database storage (NoSQL) regardless of the location of the original contents.
- **How is metadata utilized for effective searching?** The *search and find* service uses the metadata stored by the developed framework as shown in Figure 5.2. The query module of the framework is utilized for the extraction of data relevant to the requirements of users. The centralized metadata provides an efficient, user controlled and fast search mechanism for the distributed contents regardless their location in the Cloud.

.

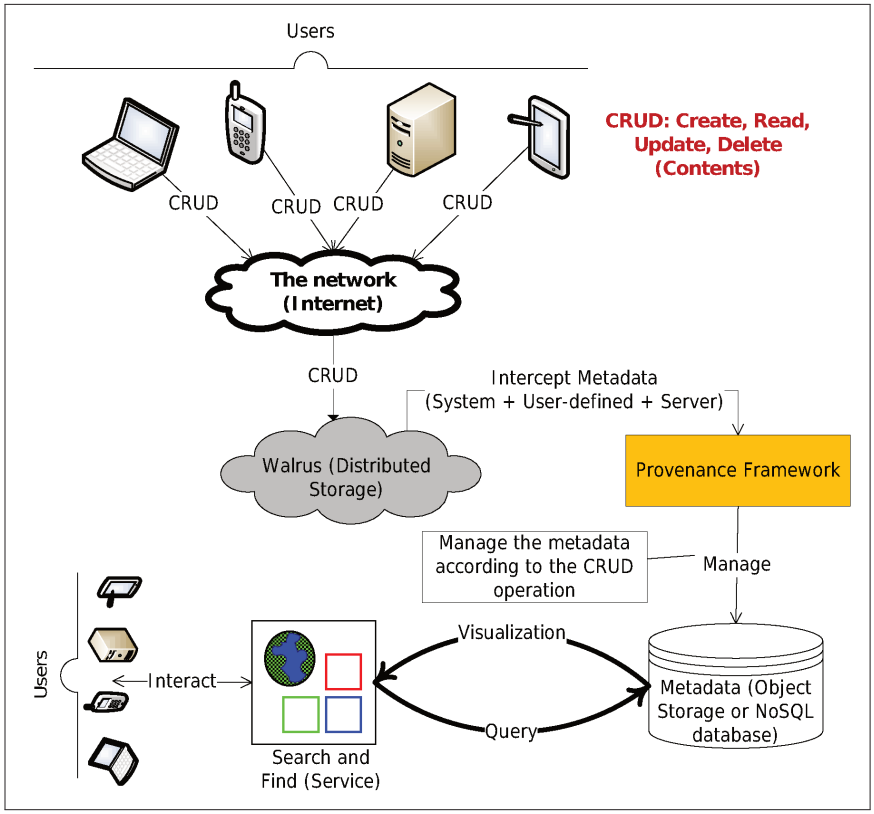


Figure 5.2: The high level architecture presenting the management and usage of metadata for content searching.

The solution to satisfy the goal of content searching is divided into the following

components:

1. **Metadata Assembler:** *How is metadata assembled for various files?*
   This component assembles the metadata associated with files which includes system data such as file name and user-defined data such as notes. The collected information are forwarded to the server module.

2. **Metadata Collector:** *How is metadata intercepted on the server side?*
   This component extracts the metadata at the server layer such as user names and their group information from the object storage service along with the metadata sent by the client. *It refers to the provenance collection module of the framework*

3. **Metadata Parser and Storage:** *How is metadata parsed and stored for efficient usage?*
   This component parses the metadata into various categories such as Access Control Policy, content types, and users information. The parsed metadata is further stored using different storage mechanisms, i.e., object storage in *XML* format and NoSQL (M/DB) schema for further utilization. *It refers to the provenance parsing and storage module of the framework*

4. **Metadata Management:** *How is metadata managed according to the different operations of object storage service?*
   This component manages the stored metadata accordingly when different operations are performed by users such as creation, deletion and modification of objects in Clouds. *It refers to the modules such as collection, parsing, storage and query of the framework*

5. **Metadata based Search:** *How the end user utilizes the search functionality for finding relevant contents?*
   This component provides a service that facilitates fast, effective and efficient *search and find* operation of contents in a Cloud by utilizing the stored metadata. For example, providing the ability to search for a list of objects that has a specific metadata value, range of values, and the support of building complex search queries with *and* and *or* mathematical operations. *This component utilizes the provenance query module of the framework*

5.1.5 Implementing the Solution: Using the Framework

We propose the following components to provide metadata based search in Eucalyptus Cloud for the Walrus service and *object storage*. These components work on the corresponding client and server modules of a Cloud.

Metadata Assembler

There are various clients that communicate with Eucalyptus Cloud for storing and retrieving data such as CloudBerry[4] and JetS3t toolkit[5]. These clients usually utilize the APIs provided by Amazon S3. Here, we used the JetS3t toolkit which is configured on a local machine to communicate with Eucalyptus Walrus. A user selects a file on the local machine and uploads it to the Cloud utilizing the client. The client architecture is altered to collect the metadata associated with files such as title and author name which is called system metadata. Furthermore, additional metadata can be added as per requirements of a user and application such as description of a file which is called user-defined metadata. The left side of Figure 5.3 depicts the normal flow where the right side presents the customized flow of the architecture of communication between the client and storage service (Walrus). The client application collects system metadata of a selected file, adds user-defined metadata, and these pieces of information are send to the server.

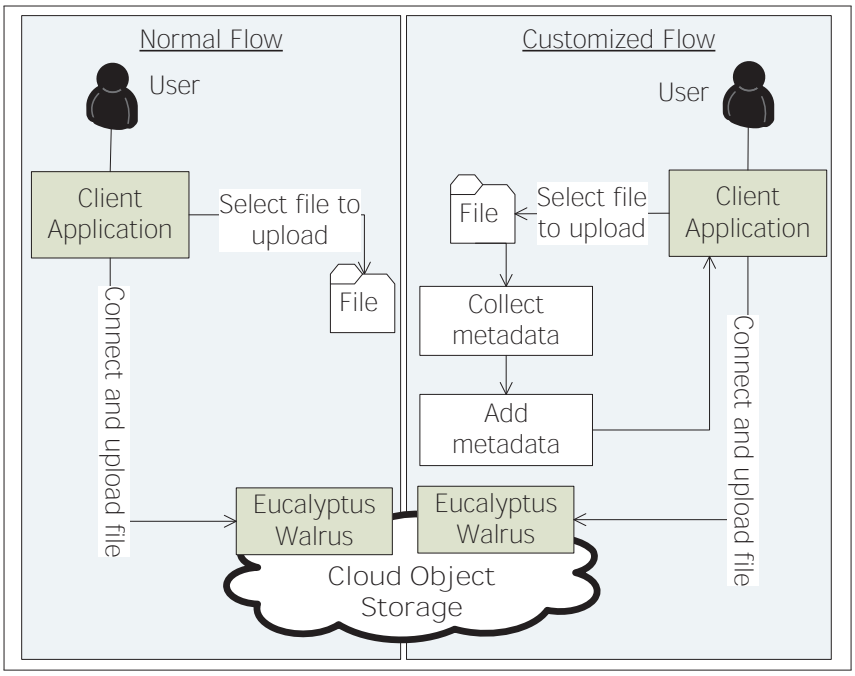


Figure 5.3: Communication between the client and storage service using a normal (left side) and the altered (right side) flow.

[4]CloudBerry: http://www.cloudberrylab.com/free-amazon-s3-explorer-cloudfront-IAM.aspx
[5]JetS3t Toolkit: http://jets3t.s3.amazonaws.com/toolkit/toolkit.html

Metadata Collector

The server side of the Cloud receives the client metadata. Furthermore, additional data such as ACP of content, owner name and group information, the location and name of the bucket and the name of created object is also collected. The additional data collected from the server is called server metadata. This information is collected by intercepting the storage service which is receiving data from various clients. Hereby, the server side metadata is augmented with the client metadata. The augmented metadata contains information, i.e., system metadata, user-defined metadata and server metadata as shown in Figure 5.4.

The additional information collected at the server is important for various reasons, e.g., to prevent the violation of security and privacy of confidential data. In this application, metadata is collected only for the contents that are described as public and available to all the users in a particular Cloud environment. Another approach could be to collect the metadata for all the ACPs and provide the search mechanisms to private and confidential data. This can be achieved by adding an additional parameter in the metadata repository that is used to authorize various users. At the moment, we do not follow this approach and provide the search for public contents. Figure 5.5 presents the architecture of the module that intercepts the server component (Eucalyptus Walrus) and retrieves various information.

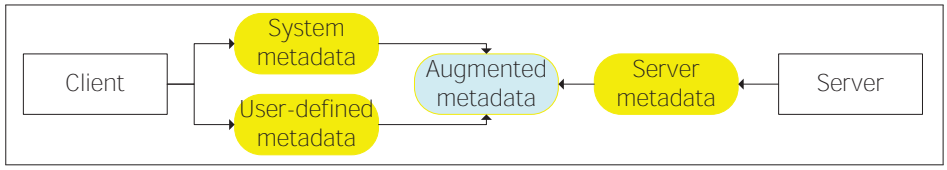


Figure 5.4: Augmented metadata which combines information from user-defined, system and server metadata

Metadata Parser and Storage

The retrieved metadata is parsed accordingly into various items and attributes. The parser module reads the augmented metadata and parses it into various categories such as content length from system metadata, client name from user-defined metadata and ACP details from server metadata along with other information as shown in Figure 5.6. Once the data is parsed accordingly, it is stored in a well-defined XML file or M/DB schema.

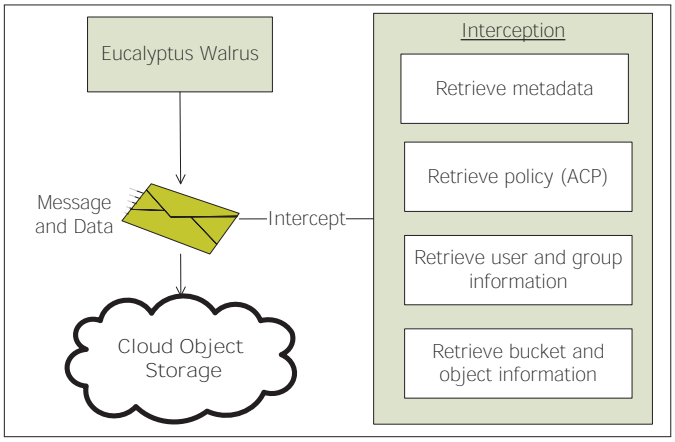


Figure 5.5: Intercepting the server component (Eucalyptus Walrus)

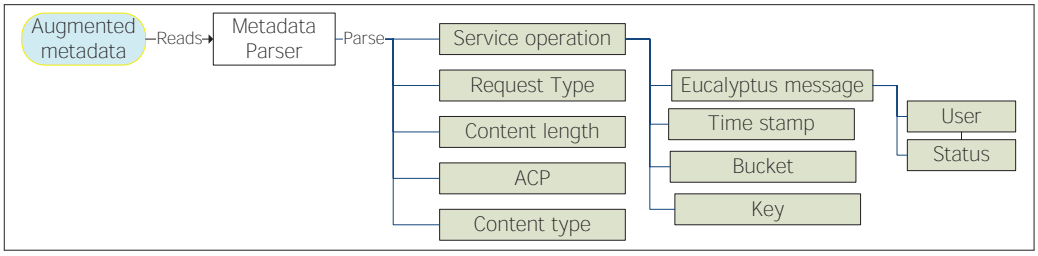


Figure 5.6: The tree structure represent the parsed metadta into various categories

Metadata Storage Mechanism

The metadata is stored using two storage protocols, i.e., XML storage (object storage) and M/DB storage (database storage) introduced in Section 4.2.4. Listing 5.1 presents a sample of the *XML* file from the parsed and stored metadata using object storage. The XML structure in Listing 5.1 represents the same tree structure as shown in Figure 5.6 for the parsed data. As shown in Listing 5.1, the root element of the XML depicts the type of operation, i.e., create contents. The root element has further child elements that depict important information stored separately such as content type, ACP information, content size, user details, and location information. These elements are stored separately to provide a fast and user controlled search mechanism where a query can be generated for individual items or their combination.

Description: Few of the elements in Listing 5.1 are described as follows. The parent element <*accessControlList*> defines the permission of any object and the child element <*permission*> depicts the allowed permission, e.g., public-read-write. Similarly, <*contentType*> describes the type of content that is uploaded which in this case is a *pdf* document. The <*euca:name*> element inside <*euca:item*> de-

Listing 5.1: XML file representing the stored metadata

```
<?xml version="1.0" encoding="UTF-8"?>
<euca:PutObjectType xmlns:euca="http://msgs.eucalyptus.com"> !-
    operation of the Walrus service, i.e., create contents-
<euca:WalrusDataRequestType>
 <euca:WalrusRequestType>
   <euca:EucalyptusMessage>
     <euca:userId>admin</euca:userId> !-user name-
     <euca:_return>true</euca:_return>
   </euca:EucalyptusMessage>
   <euca:timeStamp>2013-05-26T13:00:40.015Z</euca:timeStamp> !-time
       stamp when contents are created-
   <euca:bucket>bucket1</euca:bucket> !-name of the bucket-
   <euca:key>leung.pdf</euca:key> !-name of the object-
 </euca:WalrusRequestType>
</euca:WalrusDataRequestType>
<euca:contentLength>591748</euca:contentLength> !-size of contents in
    bytes-
<euca:metaData> !-represents user_defined metadata-
 <euca:item>
   <euca:name>jets3t-original-file-date-iso8601</euca:name> !-name of
       the client in user_defined metadata-
 </euca:item>
</euca:metaData>
<euca:accessControlList> !-represents the ACP of contents-
 <euca:grants>
   <euca:item>
     <euca:permission>public-read-write</euca:permission> !-represnets
         that contents are publicly available-
   </euca:item>
 </euca:grants>
</euca:accessControlList>
<euca:contentType>application/pdf</euca:contentType> !-type of
    contents such as pdf file-
</euca:PutObjectType>
```

scribes the client application that is used to upload the file such as JetS3t in the example. <*euca:timeStamp*> is the time when a particular object was uploaded. <*euca:userId*>, <*euca:bucket*> and <*euca:key*> provides the information about the user, the location and the name of bucket and the name of the object.

The second method utilizes the structure of M/DB to store the metadata. A domain is created that represents various metadata as described in Table 5.2. As shown in Table 5.2, various related items are clustered such as Time-Stamps and Access URLs. Non-related items such as bucket name and type of data are stored separately. Such a mechanism is designed to provide a fast, user controlled and detailed search mechanism. For instance, the clustering of Access URLs enables the search mechanism to query a single attribute-value pair.

Item	Metadata						
Attribute	Name	Bucket	Type	Size	Time-Stamps	Owner	Access URLs
Values	Readme	Software	pdf	4MB	Creation Time	Name	Http
					Update Time	Group	Https
					Modification Time		BitTorrent
					Last Access Time		

Table 5.2: M/DB domain with attributes and values for metadata storage

Metadata Management

Eucalyptus Walrus provides the functionalities to create, update, delete or access content. Depending on the particular operation, metadata is processed accordingly. For instance, the *creation* operation creates a new entry in the metadata repository but the *deletion* operation finds the existing entry and removes it accordingly. Similarly, updating any data require finding and updating the related metadata. To provide a fast, effective and reliable search mechanism, the relationship between metadata and original data is also important. For instance, creating a new data item creates a new entry in the metadata repository which contains the relationship of the user and contents, i.e., the object. Figure 5.7 presents the architecture for various operations of Eucalyptus Walrus and their connection with metadata repository.

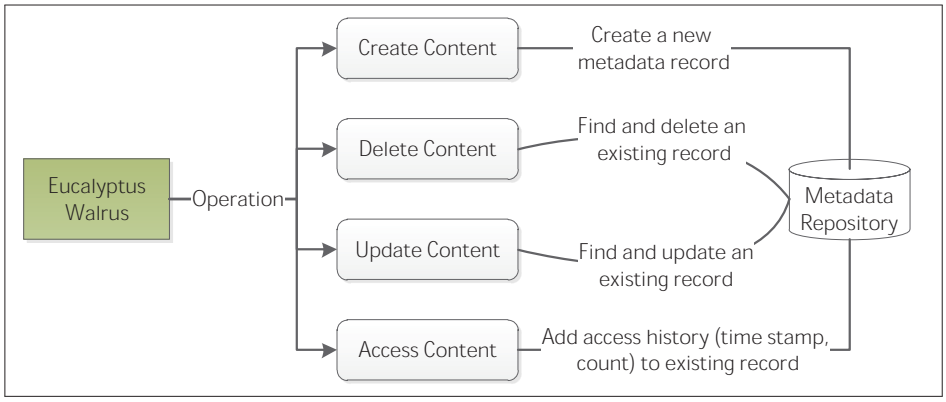


Figure 5.7: Management of metadata according to the different operations of Eucalyptus Walrus

Metadata Based Search

The presentation of relevant search results depends on the understanding of:

- **What is to be presented?** The search parameters entered by users and the relationship between the user input and the content returned as a result.
- **How should the results be displayed?** An interactive environment where end users have the opportunities to customize the results such as grouping and

sorting.

To answer these questions, we created an application (client and service) which utilizes the stored metadata according to users requirements. The detailed architecture of the application with various components is presented in the next section.

5.1.6 Metadata Based Search: Application Architecture

The high level architecture to search object storage and present relevant results is shown in Figure 5.8. The key components of the application are following:

- Client: The client takes the input from users and sends it to the server.
- Search and Find Service: The search and find service takes the input from the client and compares it to the metadata repository. The extracted results are stored in a temporary XML file.
- Interactive Display: The final results are presented in a web browser to end users where a style sheet is used for interactive display.

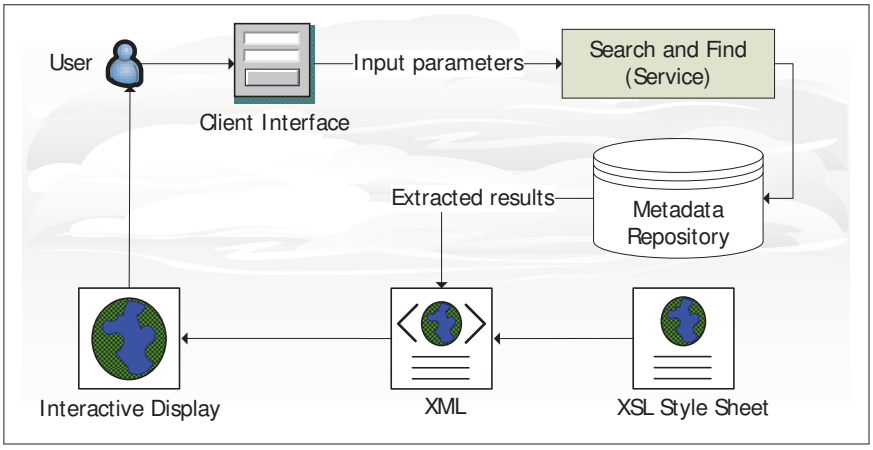


Figure 5.8: High level architecture of *Search and Find* application

Client

The client side of the application provides two kinds of search methods, i.e., *basic search* and *advanced search* as shown in the Figure 5.9.

Basic Search: In the basic search method, typing a word or phrase in the search box finds a specific object (file) in all the buckets (folders). This method searches for objects irrespective of their type and size, e.g., audio, video, spread sheet and

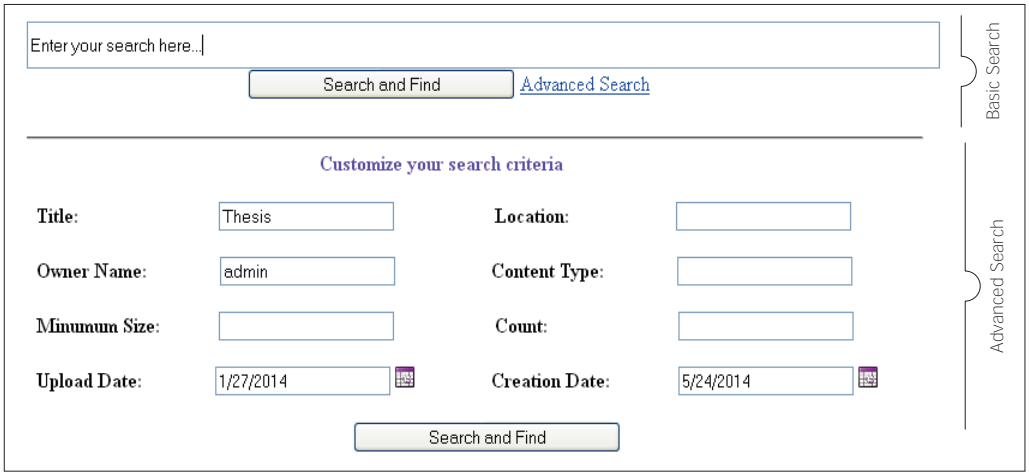


Figure 5.9: Client interface for searching content in Cloud

presentation. For instance, a user named John searches for contents with input text of 'Metadata'. The result is displayed on a web page as shown in Figure 5.10. If an item is not located, a notification appears that contents are not found with the specified criteria.

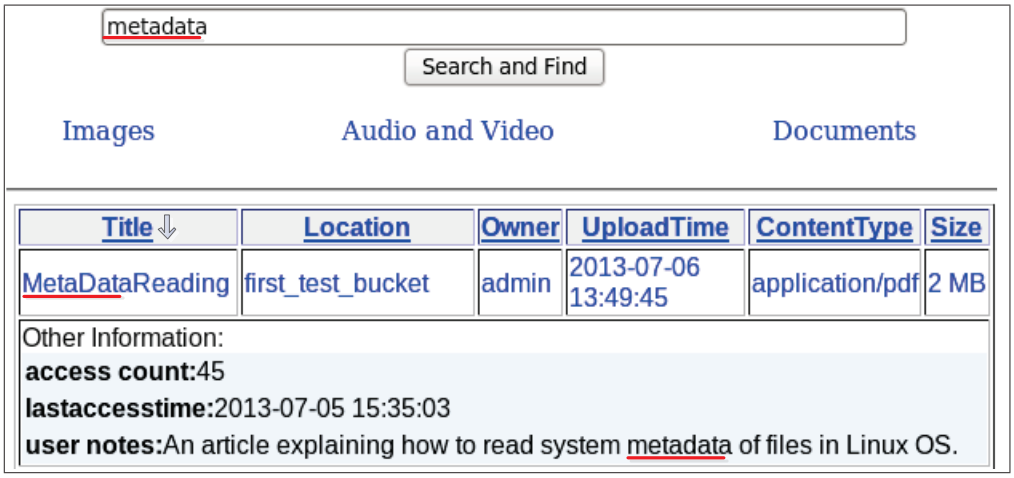


Figure 5.10: Results are displayed in an interactive fashion for the input parameters from various users. The red lines represent the matching of input parameter anywhere in the metadata such as Title (system metadata) and Notes (user-define metadata).

Advanced Search: To make the search operation effective, efficient and user controlled, narrowing the search space is possible using the advanced search as shown in Figure 5.9. For instance, when John enters search parameters of 'Metadata' as Title and 'admin' as owner, the result contains information belonging to admin where the term 'Metadata' exists anywhere in the Title as shown in Figure 5.11. Advanced

search provides the following functionalities to narrow down the search space:

- Type of document: By selecting pdf, text document, images etc.
- ACP: To select if the data belongs to a particular user, a group or publicly available in the Cloud.
- Creator and/or uploaded: If the data was created and uploaded by you (the owner) or by other users in a Cloud.
- Size: Searching for data which has a specific size limit.
- Users/Groups: To select all the documents that belongs to a particular user or a group in a Cloud.
- Custom description: User-defined metadata that could be any kind of text description of contents stored in a Cloud.

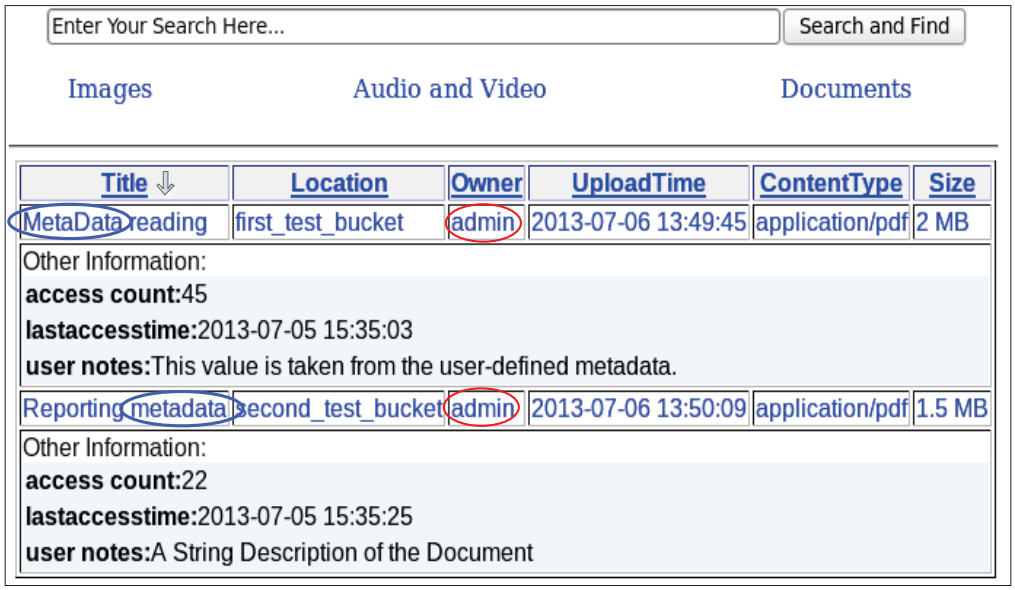


Figure 5.11: Results are displayed in an interactive fashion for the input parameters. The circles around text represent the matching of input parameters with metadata repository.

Search and Find Service

A *search and find* web service is created to search and find contents based on the metadata. The service takes the input parameters from the client. The input parameters constitute the search query that can be based on any of the basic metadata

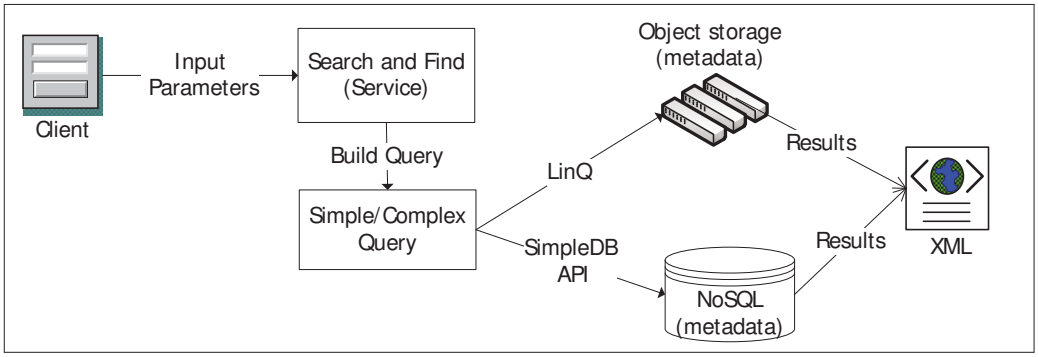


Figure 5.12: The architecture of *search and find* service which takes the input parameters and build search queries. The queries are further executed for finding relevant contents.

with the support of building complex queries. The query is analyzed by the service and the generated results are stored in a well defined XML file. This service is deployed on a virtual machine in the same Cloud where other services, e.g., Walrus is deployed.

Since the metadata is stored using two different storage mechanisms, i.e., XML format and NoSQL schema. Therefore, we also perform two different query mechanisms to search contents. When the metadata is stored in object based storage in *XML* format, we perform LINQ to *XML* technique to find content. When the metadata is stored in M/DB we utilize the NoSQL query, i.e., Amazon SimpleDB APIs to find contents. Figure 5.12 presents the basic architecture of *search and find* service.

Table 5.1 presents all the parameters that are used individually or together in finding contents. The combination of various search parameters is possible using the mathematical operators such as *AND* and *OR*. For example, the combination can be performed to find documents with a specific content type such as *pdf*, a particular owner such as *admin* and a time stamp information such as contents that were uploaded in a certain time period.

Remarks: We establish the relationship between the returned results and users input using the client interface for input and *search and find* service to build simple or complex queries. Therefore, the two components of the application satisfy the question, i.e., what is to be presented to end users.

Interactive Display

The presentation of actual results for various users input is significantly important to satisfy the question, **How should the result be displayed?** We present the

Figure 5.13: Interactive Display of results with the functionalities of grouping and sorting contents accordingly. The circles around text represents the types which are used for grouping related items.

results in an interactive fashion where users have the opportunities to customize the results according to their requirements such as grouping and sorting. For instance, John can group the results based on three different types, i.e., (i) Images, (ii) Audio and Video, and (iii) Documents as shown in Figure 5.13. The grouping of various related items increase the usability of the application for finding relevant contents. Moreover, John can sort the results based on his preferences such as Title, Access Count, Owner, Type and Size etc. Figure 5.13 further presents that items are sorted via Title as shown by the arrow key.

The display utilizes a custom XSL style sheet as shown in Figure 5.8. However, using the modularity and Independence of the components of the framework, a third party can create their own style sheets to view the results according to their requirements by replacing the existing style sheet. It is to be noted that in our current model the *search and find* web service only displays the result of the query as original or related contents. The contents are not accessible from the web page. The contents exist in the Cloud and accessibility is only possible after authentication. The authentication mechanism can be added to the web service but we have not implemented such a feature in the current model.

5.1.7 Experiment and Evaluation

The main contribution of *search and find* is the new capability of finding content added to the existing Cloud and object storage. However, it is extremely important this new capability does not impose excessive overhead. Moreover, the new capability should reveal efficient search results compared to the existing method. Therefore, the evaluation is two fold. Firstly, the measurement of execution time (elapsed time) to collect provenance and space (disk space) required to store provenance. Secondly, the performance of query protocols (for searching content), and comparing the results with the native method.

The computation and storage overhead of provenance for object storage is detailed in Section 7.3 where the Chapter 7 focuses on the provenance overhead for each layer in Clouds. The results from the Section 7.3 present that cost of provenance collection and storage is marginal both in terms of time and space. Moreover, the computation and storage overhead stay consistent regardless the size and format of various objects uploaded to the Cloud. This is satisfied using techniques such as interception, coarse grained provenance, and link based mechanism in the developed framework.

The performance of query protocols is measured through a client/server model. The server is running Eucalyptus Cloud and a virtual machine instance with M/DB deployed. The client machine is making the request to find contents based on the input parameters. The *search and find* service is also deployed in the same network and during this experiment it is executed on the same physical machine where Eucalyptus Walrus is configured. To compare the performance of queries to search for data in a Cloud, we used the following methods:

- **S3 API:** This method utilizes a native approach (AmazonS3 APIs), i.e., *hit and trial*, where the metadata of individual objects is retrieved and compared with the request parameters.
- **LinQ to XML:** This method uses *LINQ to XML* technique for the extraction of information. In this method, the XML object storing the metadata resides in the same location where Walrus stores objects.
- **SimpleDB:** This method utilizes the Amazon APIs for SimpleDB. We configured these APIs for the local M/DB database. This method provides a SELECT like feature to query the data.

Three different test cases, i.e., worst case, best case and normal case are performed to measure the performance of a sample query such as:
List the objects in Walrus where input parameter exists in the user defined metadata.
Best case means that the object to be searched is the foremost item in the metadata repository. For example, in object storage (*XML*), it is the first record. Similarly, when calling the GET method using Amazon S3 APIs, it is also the first record. In the worst case, the item to be found is the last record in metadata repository where in the normal case the searched item exists somewhere in the middle.

Table 5.3 present the performance (in seconds) of various query protocols for the different test cases when the numbers of objects are 1000 in the Cloud storage. The results in the given Table depict that LINQ is performing best for all the test cases. There are two main reasons that LINQ is performing best. Firstly, Eucalyptus

Walrus and *search and find* services are running on the same machine and there is no network communication involved. Secondly, the number of objects uploaded to a Cloud and hereby the size of the object (XML file) that is used for storing metadata. For instance, when we increased the number of objects from 1000 to 5000, the performance of LINQ is affected (degraded) as presented in Table 5.4. In real Clouds, where the numbers of objects are extremely large, the LINQ performance can be further degraded.

To handle such a situation, a versioning system can be introduced, e.g., creating a new object when the size of metadata object exceeds a particular limit or when the date is changed such as in logging mechanisms. SimpleDB APIs are performing almost the same in each case and for different number of objects as shown in Table 5.3 and Table 5.4. The reason for such a steady performance is the indexing feature provided by M/DB. The worst performance is displayed by the native GET APIs (AmazonS3) in any case presented in Table 5.3 and Table 5.4.

Query Method	Number of Objects = 1000				
	Best (s)	Worst (s)	Normal (s)	Average (s)	Standard Deviation (σ)
S3 GET API	3.977	25.793	15.322	15.03	10.91
SimpleDB API	1.219	1.228	1.226	1.224	0.0047
LinQ	0.323	0.352	0.325	0.333	0.0162

Table 5.3: Performance of query protocols (in seconds) for 1000 objects

Query Method	Number of Objects = 5000				
	Best (s)	Worst (s)	Normal (s)	Average	Standard Deviation (σ)
S3 GET API	8.372	39.227	23.253	23.617	15.43
SimpleDB API	1.217	1.226	1.22	1.22	0.0046
LinQ	1.75	1.77	1.753	1.758	0.0108

Table 5.4: Performance of query protocols (in seconds) for 5000 objects

Figure 5.14 presents the performance of various query protocols through linear trend lines when the number of objects are increased from 1000 to 5000. As shown in Figure 5.14, the average overhead of *hit and trial* method is constantly increasing with the number of objects. This clearly presents that *hit and trial* approach is not practical in the Cloud environment when the objects to be searched are very large in numbers.

The performance of LinQ also increases with the number of objects in Figure 5.14 which can be handled by introducing a versioning system (similar to versioning in logging) for the object storing metadata. However, the performance of LinQ is surpassing the native *hit and trial* method. The performance of SimpleDB is steady regardless the number of objects. Therefore, the results of the various query methods, i.e., S3 GET APIs, LINQ to XML and SimpleDB APIs clearly shows that *hit and trial* method is not practical when the number of objects to be searched are very large in number. Hereby, we conclude the storage methods (XML object and NoSQL schema) of metadata and the query protocols for searching content present faster, efficient and user controlled results.

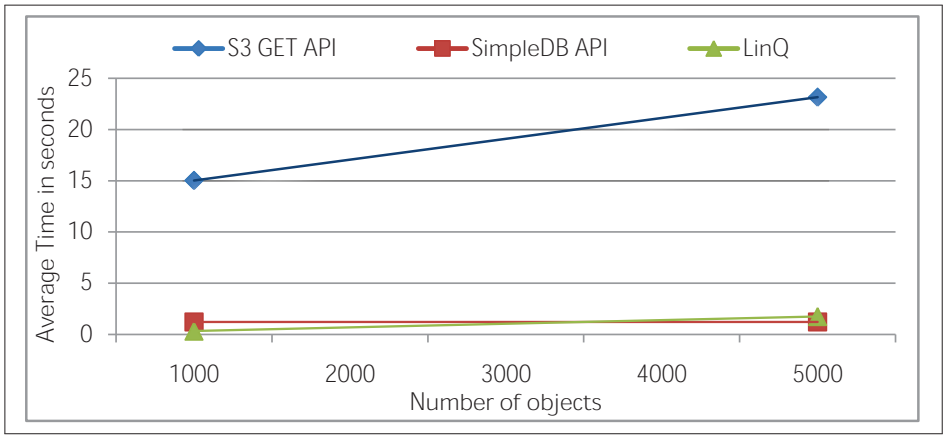


Figure 5.14: Performance of query protocols in linear fashion

5.1.8 Implication

Following are the implication and advantages of search and find contents in Clouds utilizing metadata (subset of provenance).

- **Fast, Effective and User-controlled searching:** By utilizing the metadata stored in *XML* objects or M/DB storage, searching and finding contents is much faster than the native approach of *hit and trial* method. Moreover, the storage of metadata in various categories such as type, size, and location among others provide a user controlled search mechanism. Users can choose one or more parameters or merge various parameters for finding meaningful data.

- **Consistency:** Since metadata is managed on the server component, the same consistency model is applied to the metadata that is used by the service itself.

Indeed, metadata is immediate consistent with the original model of the server component.

- **Adding Custom Parameters:** The user defined metadata can be utilized to add custom parameters. For example, adding a name-value pair of $<AccessCount>$ $<Value>$ can be used to depict the most used (famous) contents.
- **Custom Algorithms:** The metadata is managed in a separate repository. Therefore, a user can write their custom algorithms to utilize the metadata such as to move the famous contents to a faster disk unit.
- **Coupling:** Metadata is retrieved and managed on the server component of a Cloud, therefore it is consistent with the original data. Any change made to the original data will result in the corresponding change in metadata repository at the same time.

5.1.9 Summary

In this application, we provide the ability to search for contents in Clouds based on the metadata. Metadata which is a subset of provenance is managed accordingly using the developed framework for Clouds *object storage*. We utilized the components of the framework and *search and find* service for providing a user controlled, easy to use, effective and efficient search mechanism as shown by our experiments. Figure 5.15 depicts the complete architecture of the developed application which is based on the provenance framework and *search and find* service.

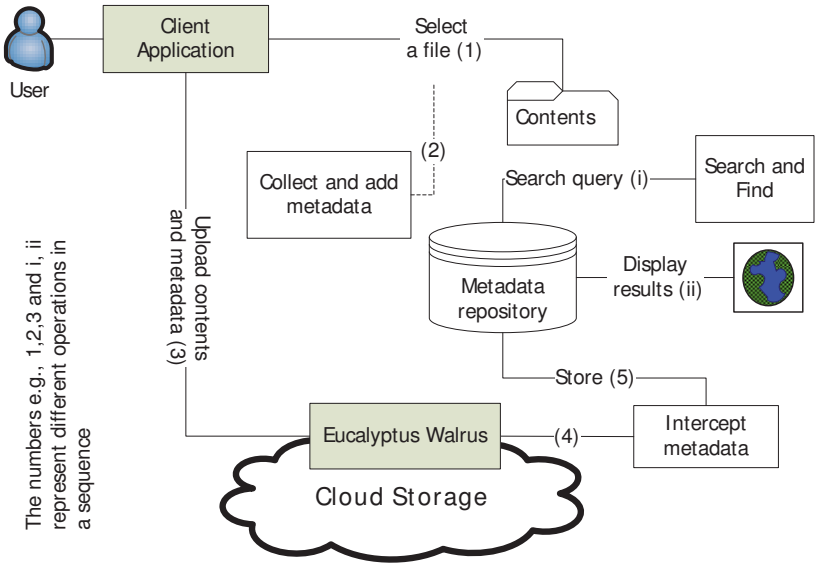


Figure 5.15: Metadata model to search and find contents in Eucalyptus Walrus

5.2 Usage Reports and Similarity Patterns in Clouds

Goal: *To utilize the provenance data of various users, groups and Clouds themselves to generate usage reports of important tasks regarding compute, storage and memory. The reports can be further utilized to find trends or similarity patterns over time in the usage of resources.*

5.2.1 Introduction

The infrastructure part of a Cloud paradigm (IaaS) offers a versatile number of resources that are distributed across a particular domain in private Clouds or geographically around the world in public Clouds. To utilize such resources, Clouds offer various features and services of compute and storage such as instances, object storage, volumes, and snapshots. These features are utilized by users, groups and Clouds themselves for various activities. The activities that are performed in Clouds take place on various Nodes, Clusters and storage resources. Therefore, the presentation such as visualization of user's activities and their relationship with Cloud services is an important aspect. For instance, presenting the computing and storage performance of various resources that are utilized by different users.

Report generation is one of the key methods to visualize various tasks in distributed computing. The generated reports are further utilized for the management of resources in distributed environments. For instance, the provider (administrator) of resources in Clouds can generate various reports of computer and storage and analyze them for finding trends or patterns in the usage from various users, groups, and Clouds themselves. Such patterns are utilized in distributed computing to provide an efficient model based on predictions.

There are various methods to find the similarity patterns, e.g., Jian Tan et. al. [125] measured the activities of an individual Node or a Cluster to find patterns in the usage of resources in distributed environments such as Clouds. Their focus is on the utilization of CPUs and the allocation of memory. Moreover, the patterns from individual Nodes/Clusters are used for efficient resource planning. Similarly, Eddy Caron et. al. [40] forecast the next request for a particular resource in a grid and Cloud environment by finding patterns in the recent history data. A prediction algorithm is used for the future requests based on the similar activity in the recent past.

Contrary to the above techniques, the developed framework provides provenance of instances, volumes, snapshots, users, groups, Nodes, Clusters and their relationships. Therefore, we propose to utilize the provenance information, collected and managed by the developed framework in order to generate enhanced usage reports

based on an individual or clusters of compute and storage services.

5.2.2 Problem Description: A Case Study

To elaborate the goal, let us consider the series of tasks performed by John in Figure 5.16. The tasks include, (i) John requests a resource with a particular instance type and submits user-data. The user-data is a custom script that deploys a web service. An instance of the resource is assigned to John, (ii) John creates a volume (block level storage) of a particular size and attaches the volume to the assigned instance, (iii) John performs some operations of the web service which generates a dataset and stores the dataset either on the created volume or directly in Cloud object storage and, (iv) if the data is stored on the volume, he creates a snapshot of the volume that can be used for later analysis or sharing with other potential users. Similar activities are performed by other users and groups in a Cloud.

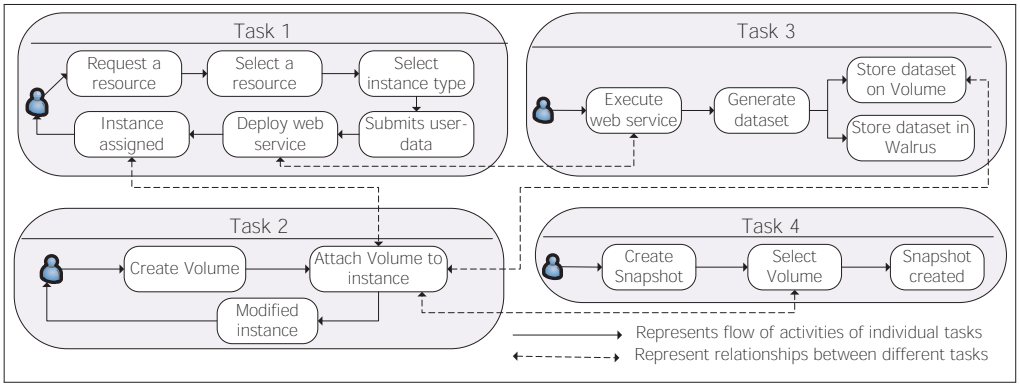


Figure 5.16: Various tasks of compute and storage performed by a user in a Cloud environment. The task number presents the activities in a sequence and the dashed lines represent the relationship between various tasks.

The actual operations of various users activities are executed using features and services like instances, NC service, CC service, and object storage among others. Therefore, we have two different perspectives for reports generation. Firstly, the user point of view, who is interested to visualize his activities performed in a sequence (the task number in Figure 5.16), and the relationship between the activities (the dashed lines in Figure 5.16). Secondly, the administrator point of view, who is interested in the performance of the provided compute and storage resources such as object storage.

The data stored in distributed Clouds is accessed by different users as shown in Figure 5.17 and the number of requests for a particular dataset can be used to define mostly utilized (famous) contents. The information from Cloud services like famous contents should be provided in a graphical format to the administrator in

order to detail the resources usage, data usage, and their behavior with users for finding trends. The analysis can be further utilized for providing an efficient model of resource planning which uses the prediction techniques and the observations from the reports. For example, to move the famous contents to a faster disk unit based on the generated report of the object storage service.

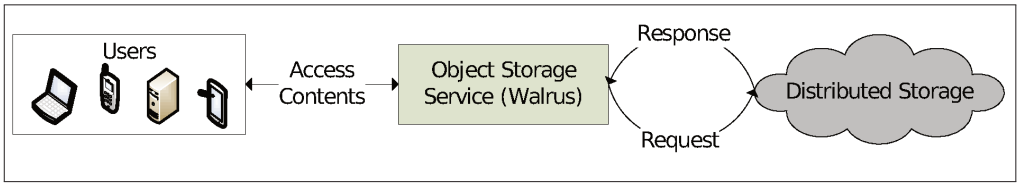


Figure 5.17: Various users interacting with object storage

5.2.3 Solution: Using the Provenance Framework

How is the framework utilized for the generation of various reports?
Figure 5.18 presents the high level architecture of the implemented solution where users interact with Cloud services on one side and report generation on the other. The provenance data, query and visualization modules are utilized respectively for finding, extracting and displaying of information according to users requirements.

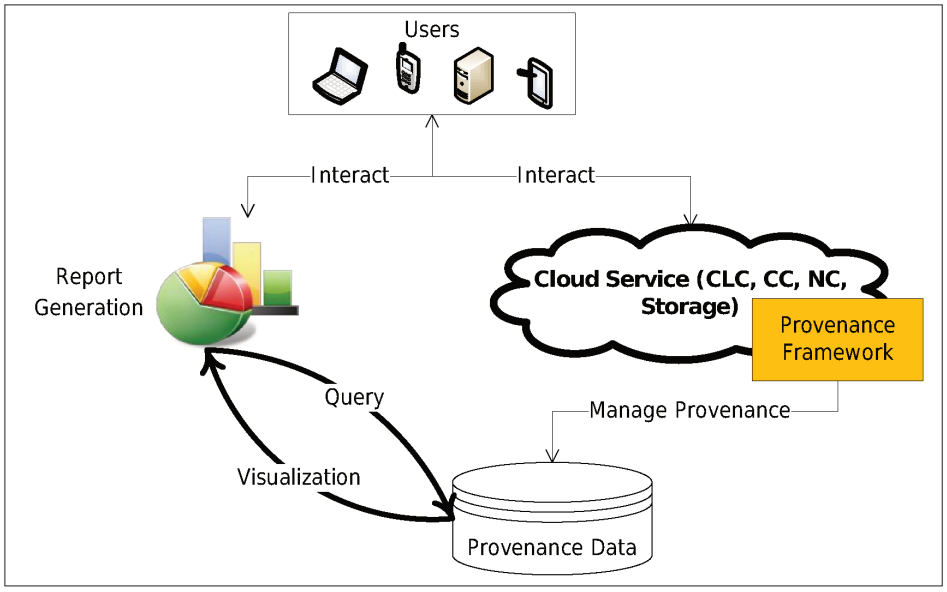


Figure 5.18: High level architecture of the usage of provenance data in reports generation.

The key components of the solution are following:

- **Client:** The client provides the interface for the selection of a time frame,

i.e., *From* and *To*, and the selection of different components/features such as users, instances, volumes and snapshots as shown in Figure 5.19. The selection constitutes the input parameters which are passed to the provenance query module.

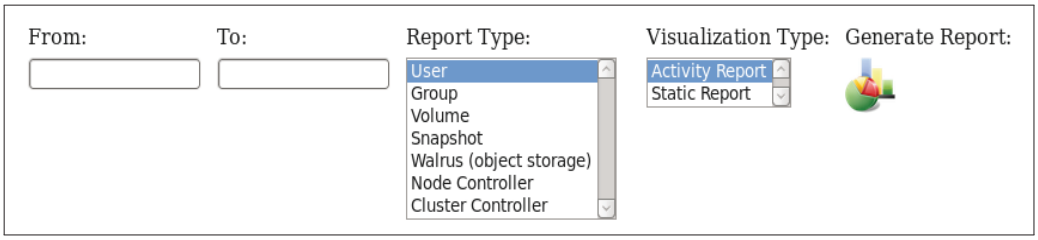


Figure 5.19: User interface for the selection of various kinds of reports.

- **Provenance Query:** The query module is used to analyze the input parameters and extract various information from the stored provenance data. The input parameters from the client provides a time frame and the component/feature which is used as a key for the extraction of information as shown in Figure 5.20. For example, if a particular time frame is selected for volumes, the result will contain information during that particular time period for all the volumes such as, (i) the time they are created, deleted, attached and detached from/with instances, (ii) the size of volumes, (iii) the instance information where the volumes are attached/detached and, (iv) the snapshots created from the volumes etc. The result of provenance query is stored in a well defined XML file which is used as an input by the visualization module for report generation.

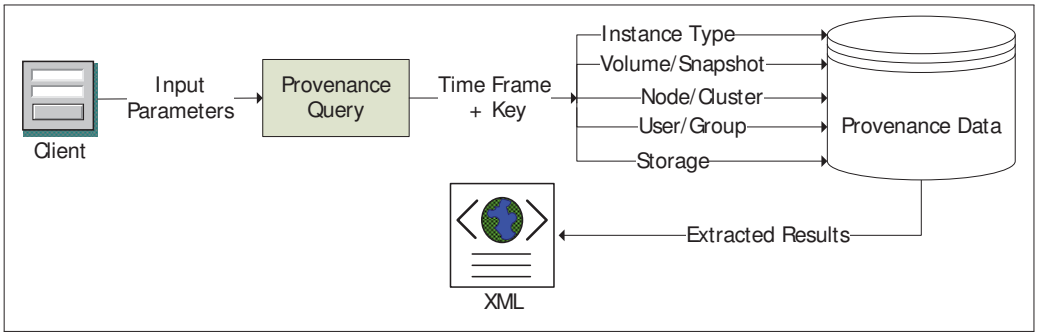


Figure 5.20: The query protocol uses time frame and input parameter as keys to extract various information from provenance data.

- **Provenance Data:** The provenance data contains the information regarding the activities of users and the activities of Clouds themselves. For instance, provenance data contains the information about users, their groups, and when

they utilized various compute and storage resources. Similarly, provenance data contains the information of the activities of Cloud services such as Nodes, Clusters and Walrus. The relationships between users and their activities with Cloud services are also maintained in provenance data via copy and link mechanism (see Section 4.2.4 for details).

- **Provenance Visualization:** The visualization module is used to present the extracted information stored in the XML file in various formats. The display of the results depends on the extracted information. For instance, the result may contain the information about the activities of a particular user, Cloud service, e.g., NC, or features of a Cloud such as Volumes and Snapshots. Similarly, the result may contain information of a particular resource such as, the number of times a resource is requested in a Cloud. Therefore, the visualization module presents the reports based on two key factors, i.e., *Activity Report* and *Static Report* which are explained next.

Reports Types

Figure 5.21 presents the basic architecture of visualizing various reports, i.e., Activity and Static reports. The Activity Report presents the tasks performed by a user, group, or Cloud services in a sequence. The relationship between the user activity and Cloud is also presented. For instance, if John performs the Task 1 (request a resource) in Figure 5.16, the report will contain the activity of John and Cloud service, i.e., the Node Controller utilized to assign an instance of the resource to John. This report is presented in web browser using XML and XSLT.

Secondly, the Static Report which presents, (i) the performance of various instances, (ii) the number of volumes and snapshots, or (iii) most used contents in a Cloud etc. This report is presented to the user in various types of charts (lines, pies and bars etc.) or tabular format. The examples of these report types are presented in the next section. Moreover, the modularity and independence of the framework provides the ability to visualize reports using any third party visual editors such as yED Graph Editor[6].

5.2.4 Experiment and Evaluation: Various Features for Report Generation

What are the key features and parameters available for the generation of various reports?

The selection of usage reports is based on the compute and storage resources, the

[6]http://www.yworks.com/en/products_yed_about.html

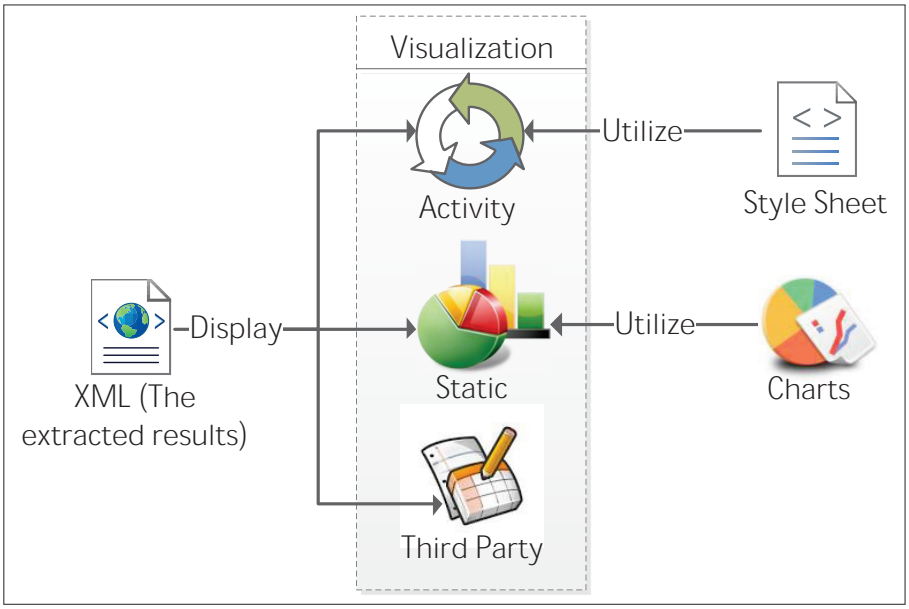


Figure 5.21: The visualization of various reports utilizing the result of provenance query in XML format.

activities of users, the activities of Cloud itself, and the relationship between the users and Clouds activities among others. Therefore, the generated reports are of various types depending on the provided features in a Cloud environment.

Users and Group Reports

The user report provides various information about the user activities and their relationship with each other and with Cloud services. This report is used to track various tasks performed by a user. Similarly, the group report provides information about group activities and their relationships with the Cloud.

For instance, John in Figure 5.16 performs the following tasks: (i) create an instance, (ii) creates a volume and attaches it to the instance, (iii) store some data in Walrus, and (iv) creates a snapshot from the volume. John wants to trace his steps for finding when he created the volume and where he attached the volume. Therefore, john selects a time frame, his user name in the Cloud and generates an activity report as shown in Figure 5.22. The report details all the steps and activities in a hierarchical format performed by John during the selected time period and the relationship of those activities with Cloud services. The circles represent the relationship between John's and Cloud's activities. The report details the necessary information regarding the volume, snapshot and instance etc. for tracking John's tasks.

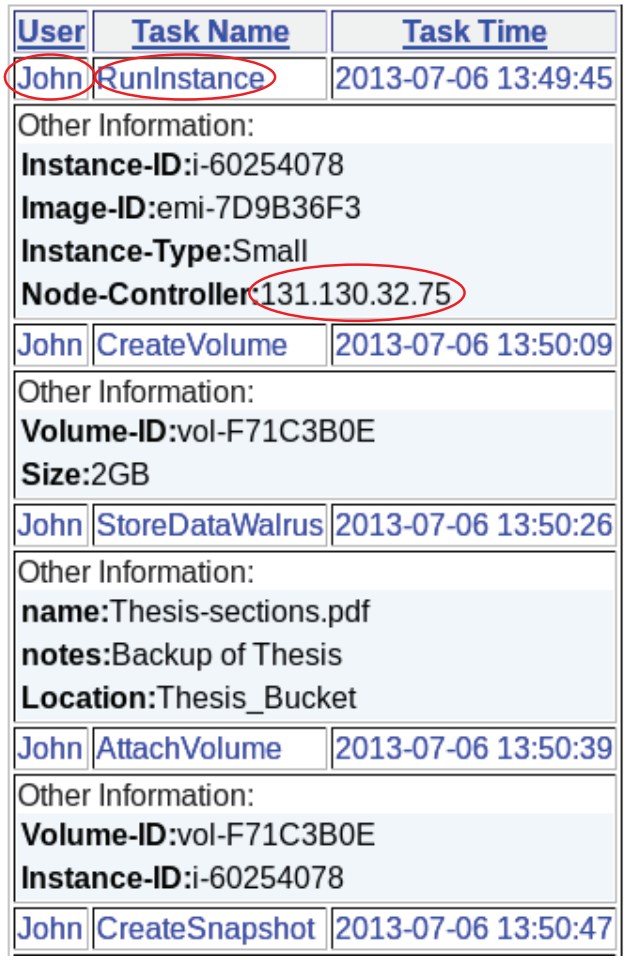


Figure 5.22: The activity report (hierarchical format) presenting various tasks performed by a user in a particular time frame. The circles around the text represent the relationship between the activities of a user and the activities of the Cloud.

Cloud Services - Nodes and Clusters

This report provides the information of the activities of various Nodes and Clusters for a selected period of time. This report can be used for efficient resource planning based on the activities of an individual Node and/or Cluster. For instance, when the administrator selects a time frame and a particular Node Controller (the same node where John performed his tasks), the result displays all the activities performed by that NC from various users as shown in Figure 5.23. The circles in the generated report present the activity of NC and the relationship with the respective user.

Node Controller	Task Name	Task Time
131.130.32.75	RunInstance	2013-07-06 13:49:45
Other Information: **Instance-ID:**i-60254078 **Image-ID:**emi-7D9B36F3 **Instance-Type:**Small **User-ID:**John		
131.130.32.75	AttachVolume	2013-07-06 13:50:26
Other Information: **Volume-ID:**vol-F71C3B0E **Instance-ID:**i-60254078 **User-ID:**John		

Figure 5.23: The activity report presenting various tasks performed by a selected Node Controller.

Object Storage

The storage report provides information such as: (i) number of contents, (ii) location of each content (iii) owner details, (iv) when and where the contents were created, (v) ACP of contents (vi) storage utilized by each user and (vii) the access count of each content etc. This report establishes the relationship of various users needs and Cloud storage. For instance, when the administrator of a Cloud generates a report for all the contents in the object storage from most utilized to least utilized, a sample of the report is presented in tabular format as shown in Figure 5.24. The report details the content title, location, owner, and the Access Count which is used to define famous (most or least utilized) contents. The administrator can utilize the report to move least used contents to a slower disk unit and vice a versa for the better utilization of the Cloud storage. It is important to note that the report in this case is presented in a tabular format. The tabular format is important when the result of a query contains a huge amount of information such as contents in object storage.

Title	Location	Owner	UploadTime	ContentType	Access Count
Reporting Features	first_test_bucket	john	2013-07-06 13:50:09	.pdf	27
Dissertation	first_test_bucket	admin	2013-07-06 13:50:26	.pdf	20
MetaData reading	first_test_bucket	admin	2013-07-06 13:49:45	.doc	16
Thesis-sections	first_test_bucket	admin	2013-07-06 13:50:39	.blg	07

Figure 5.24: The representation of contents stored in a Cloud based on their Access Count which defines the most or least used contents.

Performance Reports

The performance report provides the information according to three key features of a Cloud. Firstly, the report of resource types which provides information of each resource such as type and location, and the number of requests for each resource. This report presents the most or least utilized resources in a Cloud. Secondly, the report of various instance types which present information of instance types and their utilization from various users. This report establishes the relationship between users and the utilized instance types. This report can be used to understand the needs of instance types for each user. Thirdly, the performance report of each instance such as CPU and memory utilization. This report presents information if resources are utilized to their full extent.

For the presentation of information from a user's perspective, let's consider John in interested to measure the number of instances and their types requested by the group in which he is working. Therefore, John selects the group name and instances for the report generation. Figure 5.25 presents the result in graphical format where the Y-axis represents the number of instances, and the X-axis represents the instance types. The colored bars are used to present the utilization of various instance types by each user in the selected group. This report clearly indicates the needs of individual users via the relationship between instance types and users IDs.

The administrator can also generate a report based on the usage of resources in a Cloud. For instance, the administrator selects the resource types and generates a report showing the percentage usage of each resource based on the number of requests for each resource. The result is displayed in a pie chart as shown in Figure 5.26. The chart indicates each resource type (emi-ID) and presents the percentage usage of each resource (the number of requests) from various users.

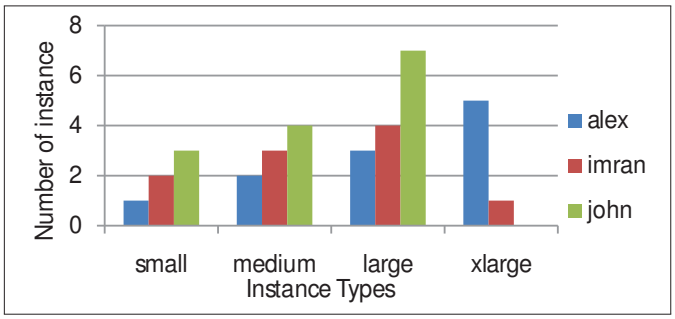


Figure 5.25: Column chart for the Cloud instance types for a particular group. X-axis presents the instance types and y-axis presents the number of instances utilized by various users.

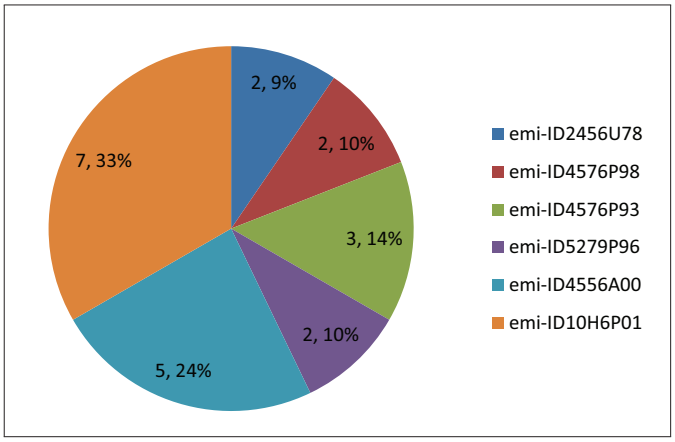


Figure 5.26: Pie chart presenting the percentage usage of Cloud resources based on the resource-ID and the number of request for each resource.

Volume and Snapshot

This report provides the information of various volumes and snapshot in a Cloud. For instance, the number of volumes, when they were created, when they were attached, where they were detached, who created them, and the size of each volume among others. The snapshot report presents additional details, i.e., the Volume ID's from which they are created. This report presents the complete status of various volumes and snapshots in a Cloud and their relationship with users and groups. Figure 5.27 presents a sample of the generated report for the Volumes created and attached by John in Figure 5.16.

Integrating the Client (Virtual Machine) Provenance

Various properties of the developed framework such as modularity and independence allow layering the provenance information from various tiers of a Cloud. For instance, we wrote a short script which measures the performance in terms of CPU and memory usage of a particular instance. This information is part of the physical layer in a Cloud. The developed framework does not collect provenance of the physical layer. However, we can merge the provenance information from the physical layer with the IaaS layer of a Cloud. The report in Figure 5.28 presents the performance of the particular instance utilized by John where provenance of physical layer is merged with the IaaS layer, i.e., instance specific information such as NC and User-ID.

Volume-ID	Task Name	Task Time
vol-F71C3B0E	CreateVolume	2013-07-06 13:50:09
Other Information: **user-ID:**John **Size:**2GB		
vol-F71C3B0E	AttachVolume	2013-07-06 13:50:39
Other Information: **user-ID:**John **Instance-ID:**i-60254078 **Instance-Type:**Small **Node-Controller:**131.130.32.75		
vol-F71C3B0E	CreateSnapshot	2013-07-06 13:50:47
Other Information: **Snapshot-ID:**s-E7ZU77G0 **user-ID:**John		

Figure 5.27: The report presenting the information of volumes, their size, creation time, attached time, and the snapshot created from a particular volume.

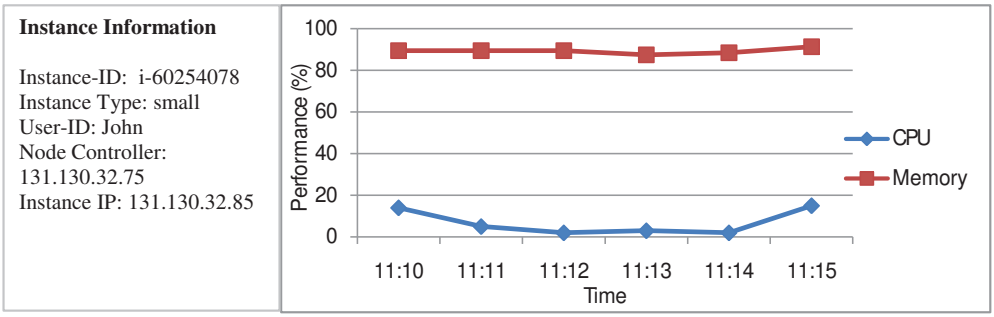


Figure 5.28: The representation of CPU and memory usage while laying the provenance from IaaS layer (instance specific information) of a Cloud.

Discussion

The above examples present a few of the report that are generated through the usage of *client*, *provenance data*, *provenance query*, and *provenance visualization* modules. However, the complete list of reports that are available may contain one or more of the features and roles presented in the Sections 2.2 and 2.4. Moreover, a user of a Cloud can find any similarity patterns based on the generated reports of various features and services. Following is a list of queries in natural language that are utilized by the framework and client application:

- Generating the report of the number of requests where EMI-ID = X (selecting a particular resource)

- Generating the report of the number of requests where EMI-ID = all (selecting all the resources)
- Generating the report of the number requests where user = admin (selecting a particular user)
- Generating the report of the number of requests where instance type = m1.small (selecting a particular instance type)

Each of the report presents a pattern in the usage of Cloud resources based on resource types, instance types or users. Moreover, multiple features of reporting can be merged to get insight details such as:

- Generating the report of the resource usage where user = admin and instance type= m1.large (selecting a particular user and instance type)
- Generating the report of the resource usage where user= admin and EMI-ID = X and Cluster Controller = $X.Y.Z.W$ (selecting a particular user, resource and CC in a Cloud)

5.2.5 Implication

Reports present the key usage points such as popular contents, highly shared data, highly used resource types and often used instance types among others. These points present similarity patterns which can be utilized for efficient planning of resources, e.g., moving the most requested instance type to a faster CPU unit. Moreover, reports present the activity and performance of the Cloud itself such as NC, CC and Walrus etc. The reporting feature provides various functionalities such as:

- **Trend Analysis:** Finding the similarity patterns in the usage of Cloud resources based on users and groups activities.
- **Data Trend:** Finding the trend in the usage of data storage and usage such as presenting the mostly utilized contents.
- **Efficient Planning:** Using the reports and analyzed trends for making reasonable predictions for future requests of resources.

5.2.6 Summary

The provenance data is utilized for generating various types of reports and hereby any patterns or trends in the usage of resources. The report contains the activities of an individual user, a group of users, a Cloud service or a particular feature of a

service. The individual module of the framework such as provenance query is utilized to analyze users requests and extract information according to their requirements. The generated reports present similarity patterns or trends based on the activity of an individual Node, a set of Nodes, a particular Cluster, a set of Clusters, and the data in object storage. The patterns are found based on users, groups and their utilization of compute and storage components such as instances, volumes, snapshots, and users-data etc. One or more of these types can be combined and therefore, the generated reports present more insight details.

5.3 Efficient Utilization of Resources

Goal: *To utilize similarities or trends in Clouds for the efficient utilization of resources.*

5.3.1 Introduction

Among various approaches, distributed computing offers VM scheduling and resource provisioning as two key techniques for resources utilization. In Clouds, the resource provisioning technique adopts a dynamic and autonomous approach for the resource allocation instead of static resource allocation. Resources provisioning requires a learning algorithm to acquire the knowledge for handling future request in an efficient manner [108]. VM scheduling of the resources is the process of load balancing and faster execution in Clouds by assigning priorities to various tasks submitted by users [77]. Green computing [24] is a term which is often used for the effective management of cost, energy and resources in Clouds and distributed computing. Apart from VM scheduling and resource provisioning, various modes such as hibernate, sleep, idle and standby are utilized to save energy in lab environments and home networks.

Contrary to the previously described approaches, we propose a method that utilizes the collected and stored provenance data for the efficient utilization of Cloud resources. As we discussed in the previous application, provenance is utilized to find the similarity patters in users requests for storage and computational resources. In this application, we focus on the utilization of similarity patterns found in provenance such as *user-data* and *instance types*.

5.3.2 Problem Description: A Case Study

In this case study shown in Figure 5.29, *user1* requests for a resource type R1, instance type N1 and submits *user-data* D1. R1 is an Ubuntu virtual machine, N1

is the small type of an instance with 512 MB RAM, 2 GB hard disk, One processor unit and D1 is a custom script that installs JAVA JDK, JRE and Apache Tomcat on the requested resource. Meanwhile, *user2* requests for a resource with the same parameters of resource type, instance type and application requirements. This kind of scenario is common in a research environment where various members work on different parts of the same application. The main components required by these users such as JAVA JDK, JRE and Apache Tomcat are same but their individual objectives are different.

In the normal or default execution of these requests, *user2* would be assigned a new resource where the required applications are installed on the assigned resource. We propose to utilize the provenance data and assign a copy of the resource to *user2* instead of creating a new resource.

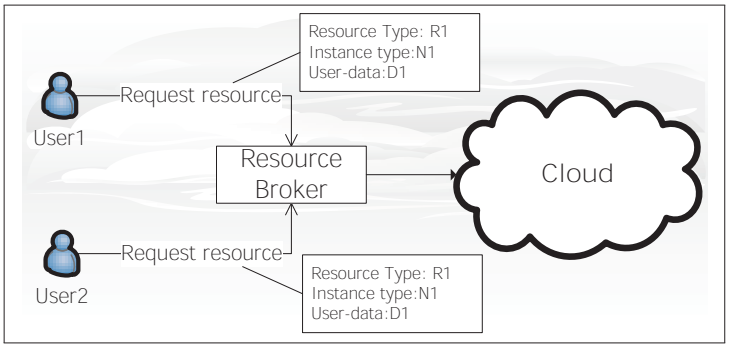


Figure 5.29: Two users requesting for the same kind of resource in Cloud computing

5.3.3 Solution: Using Provenance for the Utilization of Resources

The developed provenance framework already stores the information regarding resource type, instance type and user-data from the *user1* (presented in detail in the previous application). When *user2* makes a request for a resource, the provenance data is queried for a match. When a match is found, instead of allocating a completely new resource, an instance of the already running resource is assigned to *user2*. The generalized model for the utilization of Cloud resources through provenance is presented in Figure 5.30 where the resource allocation to *user1* is achieved with the default architecture and to *user2* with the altered architecture. Figure 5.30 also depicts the difference between the normal and altered architecture where the altered architecture utilizes the stored provenance information and provenance query modules of the framework for finding similarities in the *user-data*.

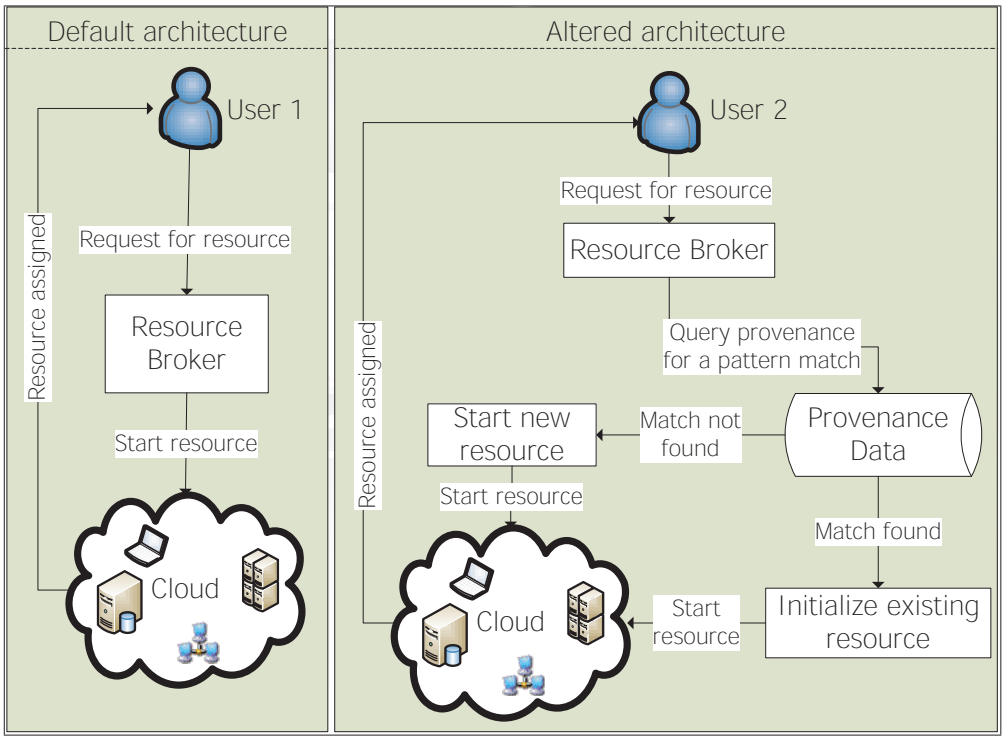


Figure 5.30: The left side presents the default architecture of assigning resource to users where the right side presents the altered architecture of resource utilization using the provenance data

5.3.4 Experiment and Evaluation

The current version of Eucalyptus Cloud does not provide the functionality of initiating a copy of a running instance but this feature is provided by other Clouds like Amazon. Therefore, we adopt a different method for testing purposes. We created a customized resource with three particular applications which are JAVA JDK, JRE and Apache Tomcat. This customized resource is uploaded to the Cloud. Another resource is uploaded to the Cloud without these particular applications. The basic architecture of both the resources is same, i.e., the same raw image (Ubuntu) is used for both the resources. Multiple instances of both the resources are requested and the execution time is recorded as given in Table 5.5

It can be noted that the average start time of the customized resource is greater than the non-customized resource by 19 seconds. Afterwards, we recorded the individual times, i.e., the download time and the installation time respectively of the three applications. We also calculated the download and installation space which is required by these three applications as presented in Table 5.6.

The time required to download and install individual applications is much higher

Image Type	Average time to run instance	Average time to stop instance	Number of executions
Image1: Populated with Apache, JDK and JRE	54 s	5 s	5 times
Image2: Raw image	35 s	5 s	5 times

Table 5.5: Average execution time (in seconds) of Cloud resources

Application	Disk space (download)	Average Download Time	Average Installation Time	Average Total Time
Apache Tomcat 7.0	8 MB	3 s	35 s	38 s
JAVA JDK	80 MB	8 s	72 s	80 s
JAVA JRE	30 MB	3 s	18 s	21 s
Grand Total	118 MB			139 s

Table 5.6: Average space (in megabytes) and time (in seconds) required by individual applications

than the time required starting another instance of the already running resource. This process saves the time required by various users and Cloud resource providers. Furthermore, the solution also saves the storage space required by individual users for their applications which are common to other users. Using time and space, we automatically save energy used by Cloud resources. The time and storage space saved by the proposed model of utilizing resources using provenance data ad similarity patterns are absolute factors. The total energy that is saved is a relative factor as described by the following formulas:

$$Time\,Saved = \sum_{i=0}^{n} TDi + \sum_{i=0}^{n} TIi \tag{5.1}$$

where TD is the time to download and TI is time to install for any component of the application. The variable i represent the number of matches that are found in provenance data. Figure 5.31 presents the total time saved for the matched application, i.e., Apache Tomcat, JAVA JDK and JAVA JRE.

$$Space\,Saved = \sum_{i=0}^{n} SDi \tag{5.2}$$

where SD is the space to download any application and the variable i represent

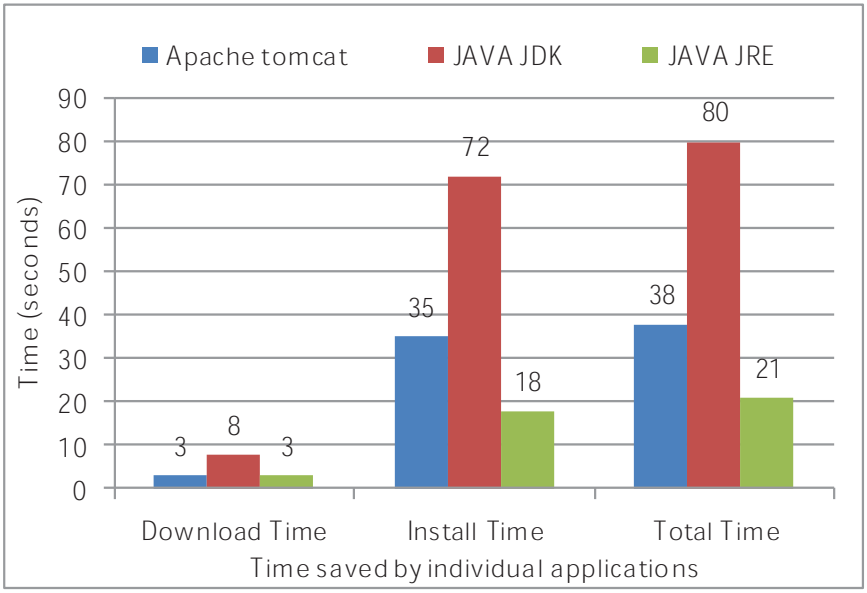


Figure 5.31: The total time saved utilizing the developed framework and provenance information that depends on the number of matches in provenance data, i.e., Apache Tomcat, JAVA JDK and JAVA JRE.

the number of matches that are found in provenance data. Figure 5.32 presents the total space saved utilizing the provenance data and existing resource for the three applications (Tomcat, JAVA JDK and JRE).

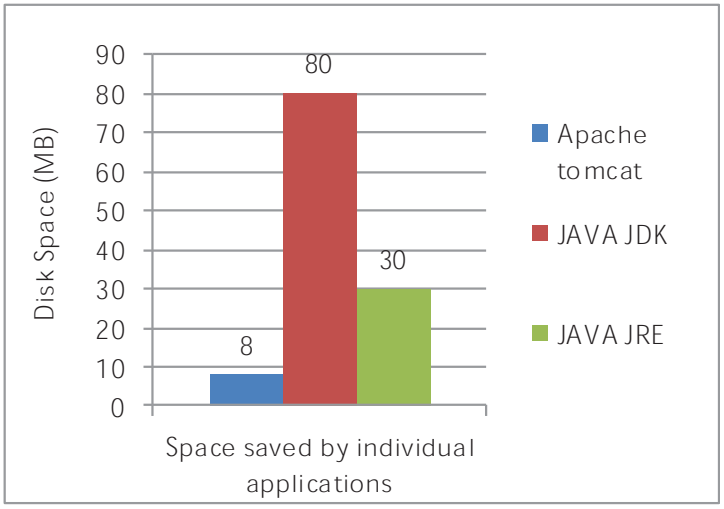


Figure 5.32: The total space saved utilizing components of the developed framework and provenance data that depends on the size of individual application.

$$Energy\,Saved = \sum_{i=0}^{n} EDi + \sum_{i=0}^{n} EIi \tag{5.3}$$

where ED is the energy required to download and EI is the energy required to install any application. The variable i represent the number of matches that are found in provenance data.

Final Remaks: The above equations and charts present that the saved time, space and hereby the energy have a direct relationship with the number of matches in provenance data. Therefore, we conclude as the number of matches increases, the efficiency of Cloud also increases in terms of saved space, time and energy.

5.3.5 Implication

Under-utilized resources accounts for a substantial energy loss in distributed computing. Clouds are designed to reduce the energy cost by using virtualization techniques and on-demand computing. However, the energy cost is further reduced using intelligent planning of resources via provisioning and scheduling techniques. Such techniques rely on the history data from resources, users, and applications like provenance. For instance, we used only one aspect of provenance, i.e., similarity patterns to save time, space and energy. Since, the framework manages the provenance data in a separate repository therefore, any third party algorithm can be applied for efficient utilization of resources according to various layers of Clouds.

5.3.6 Summary

In this application, we effectively utilized Cloud resources by reusing the existing instances. It is accomplished through provenance analysis which finds similarities such as instance type, resource type and user-data. Provenance data is used to compare a new request from users with the already running instances. The same approach can be applied for similarities on the software layer where various applications are executed.

5.4 Conclusion

Provenance is an important aspect in distributed computing and workflow applications for the verification of processes and audit trails of data. However, with e-Science centered on the Clouds paradigm in last decade, Clouds themselves have important provenance from various layers. In this chapter, we focused on the usage of the provenance data of Clouds and presented various applications where such

data is utilized. These applications present the usefulness of the provenance enabled Clouds, e.g., finding access patterns in resource usage, reports generation, provenance based search, and efficient utilization of resources. We believe, provenance is a key to make Clouds more trustworthy and reliable. Therefore, we provided compelling applications utilizing provenance and hereby satisfying that provenance enabled Clouds are more beneficial for end users and Cloud providers.

6. Aggregating Provenance from Various Layers/Service Models of Clouds

Goal: *To provide the aggregated provenance from different layers or types of Clouds paradigm for exploring the connections and relationships between them.*

6.1 Introduction

The properties of the developed framework such as modular design and independence from the underlying architectures and domains allow the collection of provenance across multiple layers of Cloud computing. Each layer of the Cloud provides significant provenance and answers to various questions specific to that layer. The applications provided in Chapter 5, signifies the importance of provenance at the object storage and IaaS layers of the Clouds. However, the aggregated provenance is important to answer questions that require provenance information across multiple layers of Cloud computing. These information include the datasets consumed and produced, different versions of services and processes, and various resources which are utilized by such services in Clouds or any other distributed and layered environment.

The aggregated provenance is used to exploit the relationships between the layers that are hidden by the abstract architecture of Cloud computing. Section 4.2.3 and 4.2.4 provides the details of layers embedded inside a particular service model, i.e., CLC, CC, NC and storage of IaaS model. In this chapter, we take a step forward and integrate the provenance across multiple layers of Cloud computing including the software, platform and infrastructure layers. The key contributions of this chapter are the followings:

- Providing the reasoning for aggregated provenance using various scenarios where only the aggregated provenance can satisfy users questions. The scenarios are demonstrated and the achievements are presented using a sample application *DataSync* which utilizes various layers of Clouds environment.

- Aggregated provenance is accomplished with the extension of the developed framework (provenance collection and parsing) to the different layers of Clouds.
- Aggregated queries are introduced and used that extract information from the various models/layers of Cloud computing, i.e., overall provenance of the Cloud.
- A fault tracking system is proposed which utilizes the aggregated provenance and aggregated query for finding silent (unknown) and non-silent (known) failures.

6.2 Understanding the Layered Architecture

The NIST [92] definition of Cloud computing divides the service models into the categories of IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Software as a Service). Cloud aware applications are developed and deployed following these various layers where, IaaS is at the bottom, PaaS in the middle and SaaS at the top. The layered approach may also include tiers such as virtualization, hardware and client tools, e.g., a web browser depending on the definition of Clouds from various business and research domains [16, 61, 82]. Complex scientific and business applications are developed, deployed, and executed using one or more of service models offered in the Cloud [3, 51, 131].

Various components of Cloud aware applications reside on different layers of the Cloud. Depending on the layered architecture, the resource providers are interested in the IaaS layer of the Cloud which supports virtualization of resources to enable computation, storage, and communication [133, 6, 101]. These resources are utilized by Cloud applications such as email service[1] and sharing documents[2], often termed as SaaS. The PaaS layer is used by developers to customize and easily develop, deploy and manage Cloud aware applications in order to fill the gap between IaaS and SaaS. The examples include salseforce [11], WSO2 [12] and providing Enterprise Service Bus (ESB) [10] as a service. Figure 6.1 presents various layers in Cloud computing and the respective view points of a user, developer and resource provider.

Each layer of the Cloud has its own provenance and generally the queries or questions to reason the collected provenance address that particular layer. Considering various layers of Cloud computing, resource providers are mainly interested in the infrastructure provenance to verify the utilization of resources through audit trials. For example, to generate reports of memory and CPU usage from a particular

[1] www.gmail.com
[2] docs.google.com

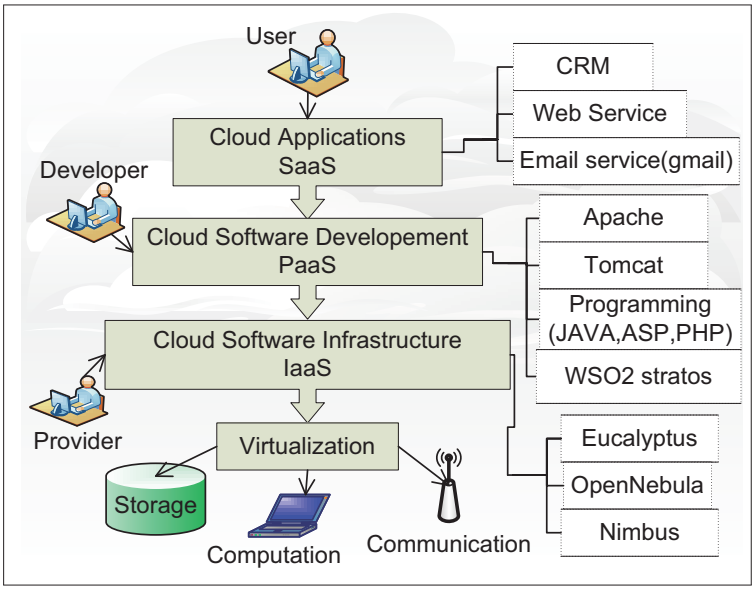


Figure 6.1: Layered architecture of the Cloud

cluster over a period of time. On the other hand, the developers of Cloud aware applications and users (researchers) are interested in the performance of deployed applications and the verification of scientific experiments. However, the aggregated provenance from various layers of a Cloud provides a unified view that exploits the relationships between the layers. This feature of aggregated provenance is significant but currently missing in mostly provenance systems.

6.3 Shortcoming of Existing Works

The existing solutions for the management of provenance data focus on a single layer of abstraction in distributed environments. For instance, application level services in workflow computing [116], workflow design specification in a grid environment [25], Service Oriented Architecture, e.g., PASOA [93] and database domains [127]. Similarly, recording system level calls for processes and files has also been addressed [96, 112, 30]. In Cloud computing, the significance of provenance data for abstraction layers such as web browsers [90], Virtual Machines using *XEN* hypervisor [88] and e-Social application (software layer) [111] has been addressed. Each of these systems provides important and significant provenance data at a single layer of abstraction. However, these existing systems cannot address queries which require an aggregated view of provenance from multiple layers of Clouds (the detail is provided in Section 3.2).

6.4 Problem Description: Why Aggregated Provenance – A Case Study

In fact Clouds are abstract and various layers are hidden from the end users, here we present an application *DataSync* depicted in Figure 6.2 to interpret these layers. *DataSync* provides two key functionalities, i.e., sync local data to the Cloud and sync Cloud data to the local machine. These functionalities are provided for data such as mails, tasks, appointments, and documents in the Cloud. Users can retrieve various data from anywhere and anytime. When users sync their data, the application checks for any new content and present them to the user. For instance, if a new task is assigned to the user, it will show up when user sync the data from the Cloud. The architecture of *DataSync* is described in the following section.

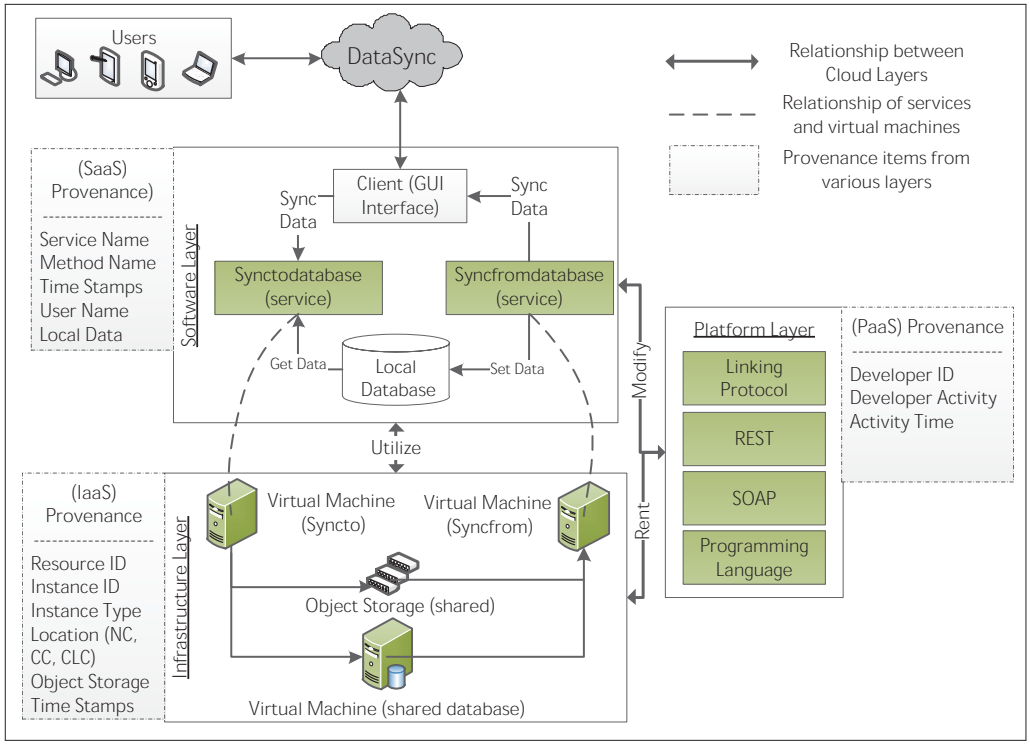


Figure 6.2: DataSync: The architecture of DataSync utilizing various layers of Cloud computing. The relationships between the Cloud layers and components of the application are also presented. For end users, DataSync is simply an application provided through Cloud for various purposes. This architecture presents various components of the application, their location and execution flow inside the Cloud.

6.4.1 Architecture of DataSync

Various components of *DataSync* reside on different layers of Clouds. The individual layers of the Cloud provide the following functionalities:

Software Layer

The software layer is composed of two components, i.e., client and web services. The client is a GUI interface used to interact with the Cloud. The client provides to view content such as assigned tasks and list of appointments for a particular user. Moreover, it provides information such as who assigned the task, meetings and appointments time, member involved and location etc. The client also provides the options to sync local data to the Cloud and vice versa to get/set the latest content.

The two web services, i.e., *synctodatabase* and *syncfromdatabase* are exposed to the end user via client. The *synctodatabase* takes the data from the client (local database) and submit or sync it to the Cloud (Cloud database and object storage). The sync process can be performed either for one particular category, e.g., appointments or for the overall data, e.g., contacts, appointments, mails and tasks etc. The *syncfromdatabase* takes the data from the Cloud and syncs it to the client (local database). Analogously to *synctodatabase*, the sync process is for one particular category or overall data.

Platform Layer

The platform layer is utilized by the developers of the application. It provides the design phase where developers choose how the services communicate with the client and Cloud. This may include linking and communication protocols such as HTTP, HTTPs, REST, SOAP, format of data exchange such as XML and text, and the design of the database including various tables etc. More importantly, this layer contains the developer name and his activities. For instance, when a developer adds a new routine to the services and uploads the routine to the Cloud, such information is part of the platform layer. The two web services of *DataSync* are deployed using SOAP protocol, Apache Axis2 web server, Eclipse IDE, and WSO2 platform for testing and evaluation purposes.

Infrastructure Layer

The infrastructure layer provides the computation and storage resources for the executions of the services and storage of data from various users. For this particular application, IaaS provides two virtual machines that host the two web services. The

object storage is used for storing persistent data such as *pdf* documents. The third virtual machine is a database server (MySQL) which hosts the database required for storing various data about mails, tasks, appointments and users.

Summary

The three service layers (SaaS, PaaS, and IaaS) of the Cloud contribute to the overall execution of *DataSync*. The software layer exposes the functionalities to the end user. The platform layer sits in the middle where developer of the services can add, delete or change the functionalities to the demands of end users. The infrastructure provides physical and virtual resources for the execution of services and storing of data. Each of the layers provides significant provenance information addressing that particular layer as depicted in Figure 6.2. The software layer provides information such as service name, user name, local data, and time stamps of user's activity, e.g., when a user synced his data. Similarly, the platform layer provides information of the developer activities such as changes made to the services. The infrastructure layer provides information about virtual machines, their type, instances ID's, their location in the Cloud and information about Cloud services such as NC, CC and object storage among others.

6.4.2 Significance of Aggregated Provenance

The *DataSync* application presents the facts that different components of the application and various layers of the Cloud share relationships or dependencies. The relationships are data and process specific such as object storage and local data, and the two services and virtual machines. Therefore, in case of failures the actual reason could be anywhere in the Cloud. For instance, if a user gets corrupted content from the Cloud, the exact reason depends on the activities of the user and Cloud, i.e., the relationship between layers. The reason of the wrong output is only clear if we aggregate provenance and expose those relationship between the layers of Cloud and components of the application. Some of the failures which require the aggregated provenance are described below.

The sync process failed - why? If a user tries to sync his data to or from the Cloud and the sync process is failed, the reason of the failure could be anywhere in the Cloud. For instance, software services are crashed, the platform is performing some update or the infrastructure resources are not responding because of the Node or Cluster failure. To find the exact reason of the failure, we require the aggregated provenance from all the layers to answer the question that why the sync process was failed.

Users are presented with wrong content: When a user syncs his data from the Cloud, and he is presented with wrong content. We require aggregating the provenance from the software and infrastructure layers in order to find the reason of wrong content. This could be either a mistake/bug in the software or the return data from the object storage is affected.
Validation of corrupted data: If a developer made some changes (add or delete a routine) to the services or database and those changes result in data corruption. We need the aggregated provenance from the software and platform layer to find out which users are affected by the changed routine. The software layer provides the information about users and input data, where the platform layer provides information about the time of added/deleted routine to the services. The combined data provides exactly which data and users are affected. For instance, data which is synced to the Cloud before the added routine is safe in the Cloud but only the local database is corrupted.
Providing the exact reason of failures in the above situations and more like them, we need the provenance from each layer of the Cloud. Therefore, we present the solution for aggregated provenance using the developed framework. We also present how the aggregated provenance solves the above problems and failures.

6.5 Solution: Extending the Framework

We extended the modules (provenance collection and parsing) of the framework to the different layers of Cloud computing to satisfy the goal of providing aggregated provenance. We utilized WSO2 platform for the delivery of the services and Axis2 as a web server. The individual steps of the solution are described in the following subsections.

6.5.1 Provenance Collection and Parsing

As shown in Figure 6.3, the collector and parser modules (provenance collection and provenance parsing) of the framework are extended towards all the layers of the Cloud. Each layer intercepts the activities performed on various parts of Cloud computing. The resultant data from the interception mechanism is passed to the parser module. The parser module gathers significant provenance according to the layer of the Cloud and present it either in XML format or NoSQL schema.

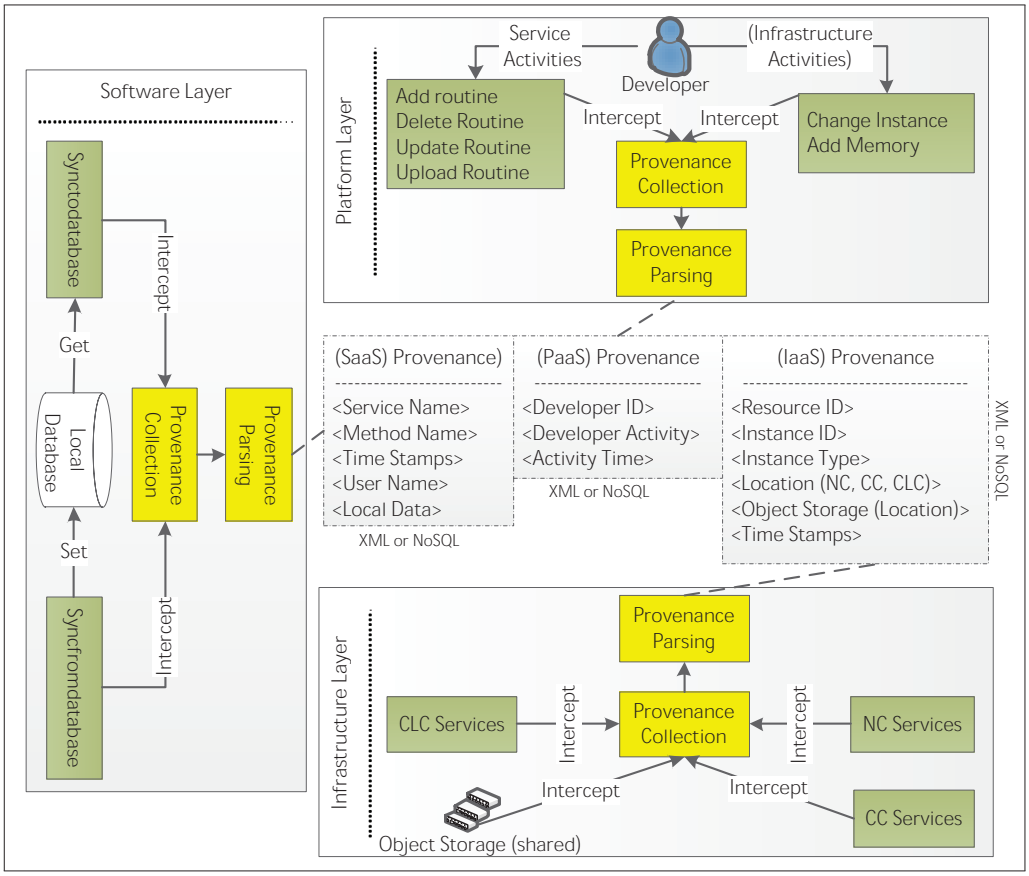


Figure 6.3: Integration of provenance collection and parsing modules of the developed framework towards infrastructure, platform, and software layers.

6.5.2 Provenance Data

Each layer of the Cloud provides significant provenance specific to the layer as shown in Figure 6.4. The software layer provides information such as service name, method name, time stamps of execution etc. The platform layer provides information about developer activities and the infrastructure layers provides information about computation and storage resources.

6.5.3 Provenance Query

The query module of the framework is utilized to extract needed information from the provenance. Since, we store the provenance of each layer separately; the concept of aggregated query is used to extract the information from overall provenance. Figure 6.5 presents the difference between a normal (the solid lines) and aggregated query (the dashed lines). As depicted in Figure 6.5, the aggregated query extracts

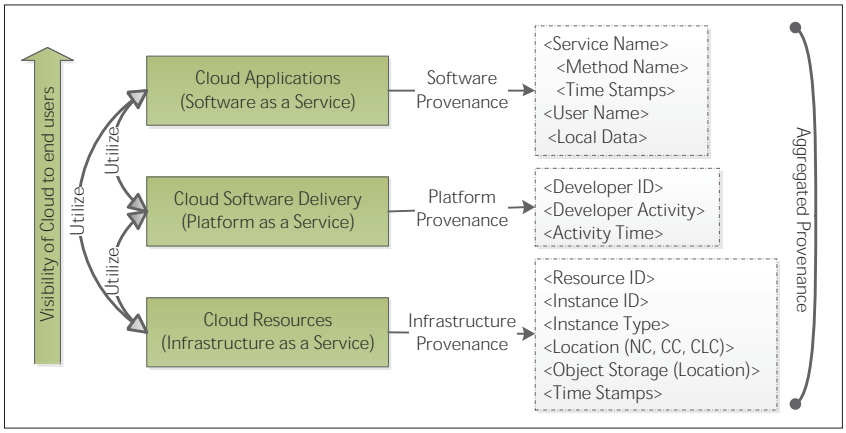


Figure 6.4: Aggregated provenance from various layers of Cloud

information from multiple layers of the Cloud in contrast to the simple query.

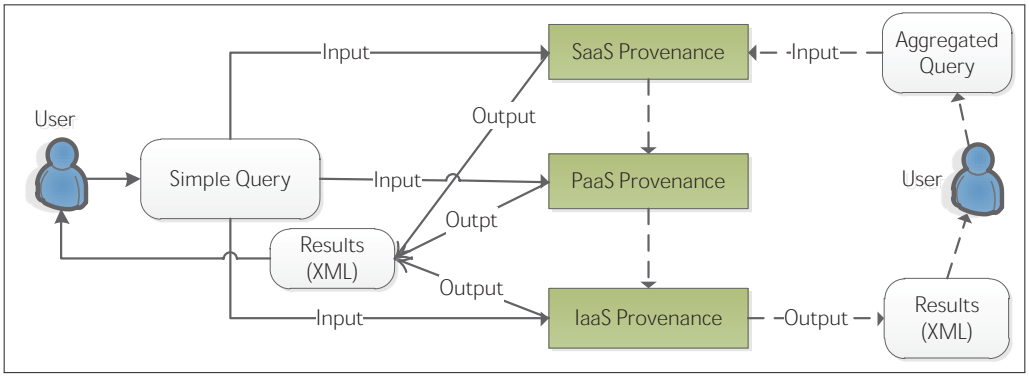


Figure 6.5: Different between the execution of simple query and aggregated query

6.5.4 Summary

Aggregated provenance is accomplished with the integration of the framework to various layers of Cloud where each layer provides its own provenance which is stored separately. Therefore, the aggregated query is used to extract the information from the overall provenance to answers various questions/problems which require information from each layer. The following section details the use-cases (problems) from Section 6.4.2 where only the aggregated provenance can provide the solution for various problems and failures.

6.6 Use Cases of DataSync

The Following use cases are explored in the context of *DataSync* which requires provenance aggregation for the resolution of problems.

6.6.1 Use-Case: The Sync process failed

Scenario:

The administrator decided for the maintenance of the Cloud. During the maintenance, something went wrong and the instance hosting the web service *synctodatabase* was terminated. The virtual machine hosting the service stopped working. During that time, John tried to sync his data but was unsuccessful.

Without Aggregation

When we examined the provenance of the software layer, we found that the sync process was failed along with other information such as time and operation that was not successful as shown in Listing 6.1. The reason of the failure is not clear from the software provenance alone.

Listing 6.1: Sample of the software provenance presenting information when the service was invoked and the return status of the service (MessageIn and MessageOut)

```
!-Ingoing message to the service-
<Time> Friday, May 30 2014 10:25 PM </Time> !-date and time of the
    activity-
<ServiceName> Synctodatabase </ServiceName> !-name of the service-
<MethodDetails> !-various details about the invoked method and input
    parameters to the method such as who is performing the task and it's
     subject etc.-
  <SyncTask>
    <name>john</name>
    <id>johd-005</id>
    <title>Doing the test</title>
    <body>This is a test for validation of failures</body>
    <assignedfrom>imran</assignedfrom>
  </SyncTask>
</MethodDetails>
-------------------------------------------------------------------------
!- Outgoing message from the service -
<SyncTaskResponse>
   <ns:return>failure</ns:return>!-return status of the service
       indicating a failure-
<SyncTaskResponse>
```

Contrary to the software layer, when we examined the infrastructure provenance, it is found the Node Controller responsible for hosting the service *snyctodatabase* was

rebooted as shown in Listing 6.2. We might suspect this to be the reason for the sync failure. However, only the infrastructure provenance is not enough to produce the evidence that the restart of the NC is the exact cause of the failure of John's sync process.

Listing 6.2: Sample of the infrastructure provenance depicting information of the NC and instance hosting the synctodatabase service

```
!- Ingoing message to the service-
<To> http://172.31.252.1:8775/axis2/services/EucalyptusNC <To> !-path
    of the node controller hosting the virtual machine-
<Action>EucalyptusNCncRebootInstance</Action> !- name of the invoked
    method at NC-
<Created>Friday, May 30 2014 10:25 PM<Created> !- time of the method
    when execution started-
<ncRebootInstance> !- name of the method and other details such as who
     and where the method was executed-
 <userId>admin</userId>
 <partition>CLUSTER01</partition>
 <name>cc_01</name>
 <ip>131.130.32.85</ip>
 <instanceId>i-AE843F32</instanceId> !-the instance hosting
     synctodatabase service-
<ncRebootInstance>
<Expires>Friday, May 30 2014 10:27 PM</Expires>!- time of the method
    when execution finished-
--------------------------------------------------------------------------

!-Outgoing message from the service-
<n:ncRebootInstanceResponse>
 <n:userId>admin</n:userId>
 <n:return>true</n:return> !- indicating that NC was successfull
     rebooted-
</n:ncRebootInstanceResponse>
```

With Aggregation

This time we combined the provenance of the software and infrastructure layer. We executed the aggregated query with the selection of a time frame when the failure occurred. A sample result of the aggregated query is presented in Listing 6.3. The result clearly indicates when John tried to sync his data, the Node Controller responsible for the instance hosting the virtual machine was in reboot phase (the second record in Listing 6.3). Therefore, the aggregated provenance establishes the relationship between the software and infrastructure layers and provides the evidence of the failure of john's activity.

Listing 6.3: Aggregated provenance presenting the information from the software and infrastructure layers together using aggregated query. The result clearly indicates the steps which are taken in a sequence for finding the exact cause of the failure

```
!- the first record in provenance presenting John's activity to sync
   his data-
<Time> Friday, May 30 2014 10:25 PM </Time> !- activity date and time-
<ServiceName> Synctodatabase </ServiceName> !- name of the service-
<MethodDetails> !-various details about the invoked method-
  <SyncTask>
  </SyncTask>
</MethodDetails>
-------------------------------------------------------------------------

!- the second record in provenance presenting NC was rebooted-
<Action>EucalyptusNCncRebootInstance</Action> !- name of the invoked
   method at NC-
<Created>Friday, May 30 2014 10:25 PM<Created> !- time of the method
   when execution started-
<ncRebootInstance> !- name of the method and other details such as who
    and where the method was executed-
 <userId>admin</userId>
 <partition>CLUSTER01</partition>
 <name>cc_01</name>
 <ip>131.130.32.85</ip>
 <instanceId>i-AE843F32</instanceId> !-the instance hosting
    synctodatabase service-
<ncRebootInstance>
<Expires>Friday, May 30 2014 10:27 PM</Expires>!- time of the method
   when execution finished-
-------------------------------------------------------------------------

!- The third record indicating john request failed -
<SyncTaskResponse>
   <ns:return>failure</ns:return> !-Return Status indicating a failure
      -
<SyncTaskResponse>
```

Discussion

In the above scenario, the aggregated provenance from the software layer and NC layer of the infrastructure provides the reason of the failure. For finding why the Node was rebooted, we can further aggregate the provenance of Cluster Controller of the infrastructure layer. The result of aggregating CC provenance with NC presents the fact of CC problem and ultimately NC restart as shown in Listing 6.4. Hereby, we conclude the aggregated provenance provides information of the exact cause of failure regardless the reason of failures such as internet lost, restart of NC, shutdown of CC, or any other problem at any layer of the Cloud.

Listing 6.4: Cluster Controller provenance presenting that CC was shut down

```
<wsa:Action>EucalyptusCCShutdownService</wsa:Action> !- method of the
    CC service invoked in Cloud-
<wsa:From>
  <wsa:Address>http://localhost:8774/axis2/services/EucalyptusCC</
      wsa:Address> !- the location where the method was invoked-
</wsa:From>
<n:ShutdownServiceResponse xmlns:n="http://eucalyptus.ucsb.edu/"> !-
    the response generated by the service-
  <n:userId>admin</n:userId> !- user who perfomed the shut down-
  <n:return>true</n:return> !- indicating that CC was shut down-
</n:ShutdownServiceResponse>
```

6.6.2 Use-Case: The sync process from Cloud presents wrong content

Scenario:

John synced his data to the Cloud while at work. Later at home he got wrong contents after syncing the data from the Cloud. The wrong output is a *pdf* file which he submitted to the Cloud earlier via attachment in an email. Something went wrong between the two sync processes and John is interested to know the reason.

Without Aggregation

We examined the provenance of software layer for the services *synctodatabase* and *syncfromdatabase* as shown in Listing 6.5 and Listing 6.6 respectively. The results indicates various information such as when and who performed the task along with other information. More importantly the provenance depicts information about content such as title and size of the *pdf* document. As shown in Listing 6.6, provenance of the item (pdf file) is changed, i.e., size and notes. However, the software provenance alone is not able to provide the necessary information why the file is changed.

Listing 6.5: Provenance information: john synched data to the Cloud

```
!-Input message-
<Time> Friday, May 30 2014 10:30
    AM
</Time> !-activity date and time-
<ServiceName> Synctodatabase </
    ServiceName> !-name of the
    service-
<SyncMailWithAttachment> !-method
    details-
  <name>john</name>
  <Attachment-title>Test</title>
  <notes>This is a pdf file for
      the test</notes>

  <size> 0.56 MB </size>
-------------------------------------

!-Output message-
<SyncTaskResponse>
   <ns:return>success</ns:return>
       !-return status indicating
       a success-
<SyncTaskResponse>
```

Listing 6.6: Provenance information: john synched data from the Cloud

```
!-Input message-
<Time> Friday, May 30 2014 10:25
    PM
</Time> !-activity date and time-
<ServiceName> Syncfromdatabase </
    ServiceName> !-name of the
    service-
<SyncMailWithAttachment> !-method
    details-
  <name>john</name>
  <Attachment-title>Test</title>
  <Notes>This is a test to view if
      the content are uploaded</
     notes>
  <size> 0.04 MB </size>
-------------------------------------

!-Output message-
<SyncTaskResponse>
   <ns:return>success</ns:return>
       !-return status indicating
       a cuccess-
<SyncTaskResponse>
```

Analogous to the software layer, when we examined the provenance of the infrastructure layer (object storage to be specific), it is found that John and another user Alex performed operations on the same content as shown in Listing 6.7. The reason for the same operation is the public read/write access to the file. However, only the infrastructure provenance is not sufficient to produce evidence that Alex operation is indeed responsible for the wrong content.

With Aggregation

We aggregated provenance of the software and object storage for the duration of time John performed the two sync processes. The aggregated query was used to extract the information from the provenance as shown in Listing 6.8. The result clearly indicates that Alex performed his operation after John, and this is the exact reason of the wrong output. Moreover, we found out that the wrong output to John is actually the file uploaded by Alex. The aggregation allows us to see the provenance of both the layers and hence the fact that content is changed. Therefore, it is possible for us to verify modifications are made to John's original data not performed by him.

Listing 6.7: Provenance information presenting the fact that john and alex synced a particular file to the Cloud with the same name (key)

```
 !- Activity of user John-
 <euca:PutObjectType> !-operation of the Walrus service, i.e., create
     content-
   <euca:userId>john</euca:userId> !-user name-
   <euca:_return>true</euca:_return> !-status-
   <euca:timeStamp>Friday, May 30 2014 10:30 AM</euca:timeStamp> !-
      time stamp when content are created-
   <euca:bucket>bucket1</euca:bucket> !-name of the bucket-
   <euca:key>test.pdf</euca:key> !-name of the object-
   <euca:contentLength>591748</euca:contentLength> !-size of content
      in bytes-
   <euca:metaData> !-represents user_defined metadata-
     <notes> This is a pdf file for the test </notes>
   </euca:metaData>
   <euca:contentType>application/pdf</euca:contentType> !-type of
      content such as pdf file-
 </euca:PutObjectType>
-------------------------------------------------------------------------

  !- Activity of user Alex-
 <euca:PutObjectType> !-operation of the Walrus service, i.e., create
     content-
    <euca:userId>ales</euca:userId> !-user name-
    <euca:_return>true</euca:_return> !-status-
    <euca:timeStamp>Friday, May 30 2014 11:30 AM</euca:timeStamp> !-
       time stamp when content are created, noted that alex performed
        the operation after john-
    <euca:bucket>bucket1</euca:bucket> !-name of the bucket-
    <euca:key>test.pdf</euca:key> !-name of the object-
    <euca:contentLength>44568</euca:contentLength> !-size of content
       in bytes-
    <euca:metaData> !-represents user_defined metadata-
      <notes> This is a test to view if the content are uploaded </
         notes>
    </euca:metaData>
    <euca:contentType>application/pdf</euca:contentType> !-type of
       content such as pdf file-
  </euca:PutObjectType>
```

6.6.3 Use-Case: Corrupted Data

Scenario

A developer decided to add a routine to the services which submits and extracts data to/from the Cloud using an updated driver. He also updated the drivers on the virtual machines to comply with the added routine. The driver has bugs and the data send or retrieved by users got corrupted. We want to know exactly which users and data is affected by the added routine.

Listing 6.8: Aggregated provenance from the software and infrastructure layers depicting activities of the two users with the content having same key

```
!-Record 1 in provenance: when john synched his data-
<Time> Friday, May 30 2014 10:30 AM </Time> !-activity date and time-
<ServiceName> Synctodatabase </ServiceName> !-name of the service-
-----------------------------------------------------------------------
!-Record 2 in provenance: When object storage created the content-
<euca:PutObjectType> !-operation of the Walrus service, i.e., create
    content-
<Time> Friday, May 30 2014 10:30 AM </Time> !- when the content was
    created-
<john> !-information about the user John
<metadata> !- information about his content-
-----------------------------------------------------------------------
!-Record 3 in provenance: a response for successful creation of content
     to john-
-----------------------------------------------------------------------
!-Record 4 in provenance: when Alex created his content with the same
    name (key)-
<euca:PutObjectType> !-operation of the Walrus service, i.e., create
    content-
<Time> Friday, May 30 2014 11:30 AM </Time> !- when alex replaced the
    content-
<alex> !-information about alex-
<metadata> !-information about the created content which differs from
    john's content-
-----------------------------------------------------------------------
!-Record 5 in provenance: When john synced his data from the Cloud-
<ServiceName> Syncfromdatabase </ServiceName> !-name of the service-
```

Without Aggregation

The platform layer provides information about the time and added routine to the services, i.e., when the service was updated. The software layer presents information about various users and their input/output data which they send and retrieved to/from the Cloud. Similarly, if the new routine is utilizing object storage from the Cloud then IaaS layer provides information about the affected content. Each of the layers provide important provenance however, we need the aggregated provenance to find which data and which users are affected by the added routine in the web services.

With Aggregation

When we aggregate the provenance of various layers, we can exactly identify which users got affected by the added routine. This is accomplished using platform provenance for finding when the routine was added, and the software provenance for finding users and data utilizing the newly added routine. Similarly, the infrastructure layer (object storage) presents information about content uploaded with the

newly added routine. Moreover, the aggregated provenance also identifies if the data in the Cloud is corrupted or not. For instance John is getting corrupted date when he synced his data from the Cloud. However, when john synced data to the Cloud, it used the service before any changes. Therefore, aggregated provenance also validates, John's data in the Cloud is not corrupted but only the extraction of data is wrong.

6.6.4 Discussion

The above use cases present the significance of aggregated provenance for finding various failures and the reason of those failures. Moreover, aggregated provenance is utilized for validation and verification corrupted content, users activities and wrong outputs. The aggregated provenance identifies the source of various failures and who is affected since the failure occurred. Therefore, aggregated provenance is important for solving various problems at different layers of Clouds. In the next section, we present how the aggregated provenance is utilized for building a fault tracking system in Clouds.

6.7 Fault Tracking (Error Handling) in Clouds

Clouds use a diverse set of resources, applications, hardware, and network protocols for the delivery of various services. Research Clouds in particular utilize open source libraries for communication between various components, web services and storage mechanisms, e.g., Axis2/C, Apache and Mule [10] in Eucalyptus. Moreover, unexpected failures such as software crashes are part of any distributed system. Therefore, Clouds in general and research Clouds in particular are prone to errors and faults. To resolve such failures, a fault tolerant system is used in distributed environments.

A fault tolerant system is mainly composed of two parts, i.e., fault tracking (detection of faults) and fault handling (corrections of faults) [64, 41]. Fault tracking is the process of finding a fault and logging the result or showing a proper message to the user. Fault handling on the other side is the analysis of a current failure by mapping the data from recent events which might caused the failure, thus finding the root cause of a failure [64]. Once the root cause of a failure is analyzed, the proactive or reactive approach [135] is taken into consideration to resolve the failure. The various steps required in a fault tolerant system are shown in Figure 6.6.

Proactive fault handling [100] is the process of prediction of upcoming failures. This involves the analysis of the root cause and making the necessary decisions to resolve the failure [110]. Reactive approach on the other hand, utilizes the exception

raising technique to resolve failures. For complex and scientific applications such as workflow execution (SOA architecture), failure handling can be further divided for individual tasks and layers as proposed in [70]. Individual task failure may be handled by various approaches like restarting the task, putting checkpoints at various levels and making a copy of the original task [17, 63].

It is important to note that the implementation of a fault management system requires an extensive knowledge of each fault and methods to resolve them. This can vary according to the domain, application and recovery mechanisms [64]. This thesis does no cover the actual management of faults but instead provide the data that can be utilized to build such a system.

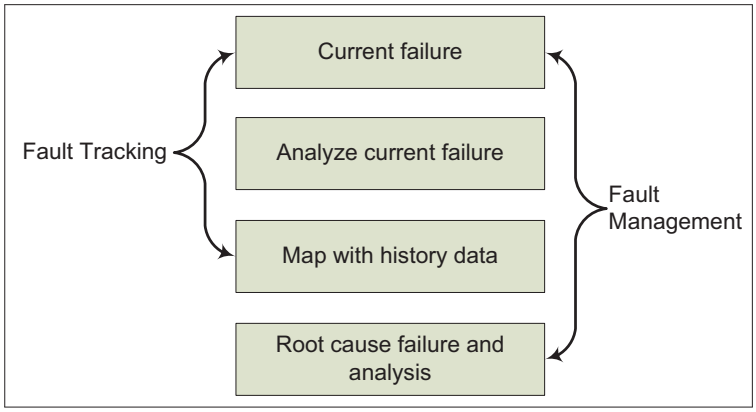


Figure 6.6: Fault tracking and fault management

6.7.1 Fault tracking using provenance

Failures in distributed systems are categorized in two types, i.e., silent (unknown) and non-silent (known) [32]. For instance, if we call *synctodatabase* service with wrong arguments, this will cause a failure which is known to the system. Such failures are detected and stored by the provenance framework as shown in Listing 6.9. The detail of provenance configuration for known failures, i.e., faulty flows is provided in Section 4.5.3. The silent failures occur when something goes wrong during processing of the request and the root cause of the failure could be anywhere in the distributed system. In the above Section 6.6, we presented various use cases of different failures such as wrong output and corrupted data. Such failures require an intensive investigation for finding the root cause of failures.

In distributed systems such as Clouds, fault tracking is mainly accomplished by logging errors into text files. These logs contain *warning*, *info* and *error* messages. However, logs are limited and lack the semantic meaning of errors unlike provenance. For instance, the relationships between various events of Clouds and the respective

Listing 6.9: Presentation of a known or non-silent failure using provenance

```
!- the presentation of a know failure in provenance-
<Fault> !- indicating that a fault has occured-
  <Code>
    <Value>Receiver</Value> !- indicating that fault occured on the
        recieving side of the service-
  </Code>
  <Reason>
    <Text>wrong number of arguments</Text> !- indicating the reason of
        the failure-
  </Reason>
</Fault>
```

logs are not established. More precisely, provenance contains information where logs contain data or messages. Contrary to logs, provenance is useful in finding the root cause of any failure through investigation of recent activities.

The aggregated provenance records the relationships between the processes of Clouds, users activities, and various data items. These relationships are exposed through aggregated query when something goes wrong in the system. More precisely, the result of an aggregated query presents users activities and the resultant Cloud activities. Therefore, aggregated provenance plays a key role in finding the exact cause of failures. The combination of silent and non-silent failure using the developer framework provides the opportunity to track various failures, and find their origin. Moreover, it provides a list of users and data items which are affected. Figure 6.7 presents the high level architecture of a fault tracking system utilizing provenance of Clouds.

The provenance of particular layer highlights any failure and the aggregated provenance track the failure to its origin. Therefore, fault tracking and management is enhanced with the collection of related information on other layers of the Cloud as shown by the aggregated query in Figure 6.7. Fault tracking utilize various information from Clouds layers such as different processes, invoked methods, input/output data, resources utilized, and changes made to the resources and services etc. Using provenance as a means of finding root cause of any failure has the following advantages:

6.7.2 Advantages

- **Related data:** The framework captures and provides the related data for any event. For instance, the input and output parameters when a service is invoked. These parameters are important for finding the exact reason of a failure.

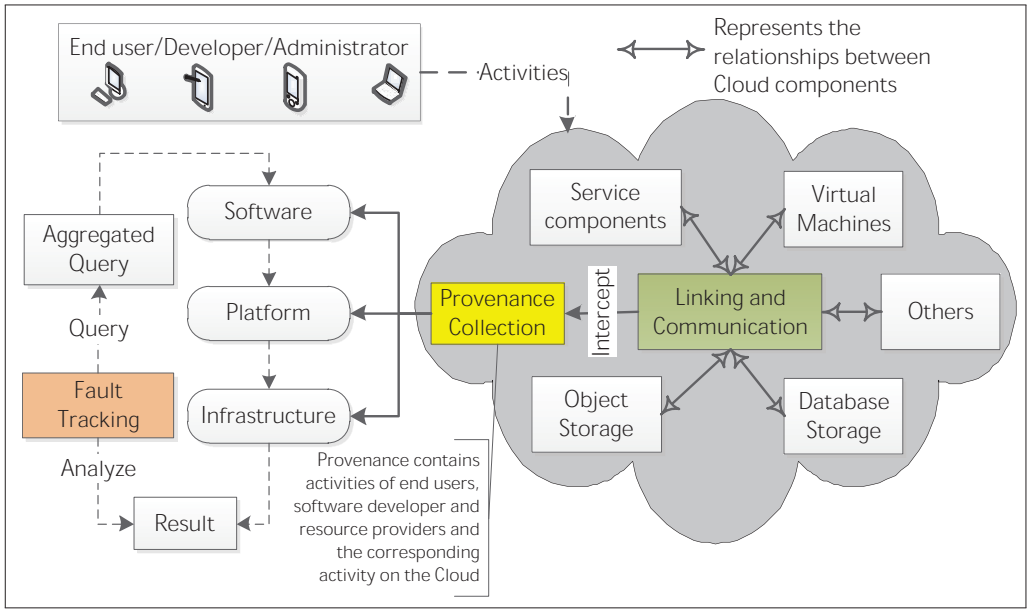


Figure 6.7: Fault tracking using the collected provenance and aggregated query

- **Provenance is event based:** Provenance supply information according to the activities of users and the corresponding activities of the Cloud anywhere in the distributed system. Therefore, in case of failures, provenance provides the recent activities for analysis.

- **Reduced effort:** The well defined structure of the stored provenance allows easy understanding and query of the information. Therefore, the effort to find the root cause of any failure is reduced using a well-managed event based provenance data.

- **Custom messages:** The provenance of various components is collected using the developed framework. Therefore, users can add custom messages for various failures since the framework support modularity and independence.

- **Controlling the amount of provenance:** The user has a complete control over the collection of provenance across various layers of Cloud. Therefore, the framework allows configuring provenance to a particular service, a method, or a component.

Limitation: Unfortunately, each version of the Eucalyptus Cloud is deployed using the *Mule 2.0.2* version of the framework. This particular version does not support interceptors. The interceptor concept was dropped for this version while the previous and later versions of Mule provide the support of interceptors. Therefore, we changed

the source code of Eucalyptus Cloud and built the Cloud with *Mule 2.2.0* libraries instead of using 2.0.0 to prove the concept of error handling using interceptors. The change was not completely successful because of the change in new libraries and their communication mechanisms. Therefore, we were not able to generate fault messages for the CLC components to its full potential.

6.7.3 Summary

Distributed computing such as Clouds follow SOA architecture where the executions of events utilize a diverse set of resources. Therefore, in case of failures, it is important to present the actual failure and provide the related data. The combined data, i.e., actual failure and related information are analyzed for finding the root cause of failures. Moreover, corrective actions can be taken to resolve the failure such as in fault management systems. Such a system can be developed on top of our fault tracking system which already provides the necessary information to analyze the faults.

6.8 Conclusion

Cloud computing follows a layers architecture namely *software*, *platform* and *infrastructure*. On one side, each layer of the Cloud provides significant provenance which generally target the audiences at that particular layer. On the other side, the aggregated provenance is the result of accumulating provenance from individual layers which exploits the relationships that exists between various layers of distributed computing such as Cloud. In this chapter, we put emphasis on the aggregated provenance from various layers or types of Cloud computing. Firstly, we highlighted the importance of aggregated provenance using a *DataSync* application and various scenarios which require the aggregated provenance. Secondly, we integrated the developed framework to the software, platform and infrastructure layers for the collection and management of provenance. Hereby, we explored the concept of aggregated query which extracts information from provenance of all layers. We explored various scenarios from *DataSync* application where the answer to a specific problem is only possible if we have the aggregated provenance. We also provided that known and unknown failures can be resolved through investigating the aggregated provenance, i.e., fault tracking and management. The architecture of the developed framework proved to be versatile and provide the ability to integrate provenance collection across multiple layers of Clouds.

7. General Evaluation of the Provenance Framework

It is generally believed as mentioned by leading computer scientists that every computer science problem can be solved by imposing an extra layer of indirection [121]. However, the extra layer cost some overhead. The integrated solution of provenance in Cloud computing, particularly for the service models such as *infrastructure*, *platform*, *storage* and *software* causes extra overhead of computation and storage.

The computation overhead is the extra time required for intercepting, parsing and making the connection to store significant provenance data in the developed framework. To provide a solution for problems which involve storing data such as provenance, it is also important to provide strategies where the storage overhead is kept minimal. Moreover, the data should be stored in such a manner, i.e., the storage mechanism that facilitates efficient query. Furthermore, the evaluation of the developed framework involves a proof of concept that the framework is extensible to different service models with minimal changes and efforts.

We performed various test cases in order to provide the performance of the developed framework including the collection, storage and query overhead. These test cases are:

- Performance overhead of the framework for IaaS services such as *Node Controller* and *Cluster Controller* using Eucalyptus Cloud.
- Performance overhead of the framework for PaaS model such as *Application Server* and *Enterprise Service Bus* among others using WSO2 platform.
- Performance overhead of the framework for SaaS model using the web services *synctodatabase* and *syncfromdatabase* developed in the DataSync application.
- Storage overhead of the framework for provenance data which includes different protocols of storing data such as object storage in *XML* format and NoSQL schema with M/DB database.
- Query performance of the stored provenance data in different storage units and formats.

7.1 Evaluation of the Framework for Cloud IaaS Model (Using Eucalyptus)

The developed framework is tested and evaluated using Eucalyptus Cloud set as IaaS model. The Eucalyptus Cloud uses two middlewares, i.e., Axis2/C and Mule for the communication and linking between services and components. Therefore, we evaluated the developed provenance framework based on these two middlewares.

7.1.1 Performance of the Framework for CC and NC Services Using Axis2/C

In this test case, we evaluated the *Cluster* and *Node Controller* services of Eucalyptus Cloud. To get the physical evidence of computation overhead, 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 includes the provenance module for *Inflow* and *Outflow* phases of Apache Axis2/C. Evaluation was performed using the underlying architecture detailed in Table 7.1. The details of the physical machines that were used to install and run IaaS Cloud are given in Table 7.2.

Table 7.3 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, i.e., 50 executions. The average time presents the overhead for collecting, parsing and storing of provenance data into properly defined *XML* files. The results were very low (in milliseconds) for collection module of the provenance framework as shown in Table 7.3.

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

Table 7.1: Underlying architectural components used to evaluate Eucalyptus Cloud.

7.1.2 Performance of the Framework for CLC Services Using Mule

The CLC services of Eucalyptus Cloud are deployed using Mule 2.0 framework. This particular version of Mule framework does not support interceptors and therefore, we built Eucalyptus with Mule 2.1.2 libraries. However, we were not able to successfully implement Mule 2.1.2 libraries to all the components of CLC services. Therefore,

Machine No	Services Installed	CPU	Memory	Disk Space
Machine 1	CLC, CC, Storage Service	Intel Core (TM) 2: CPU 2.13 GHz	2 GB	250 GB
Machine 2	Node Controller	Intel Core (TM) 2: CPU 2.13 GHz	2 GB	250 GB

Table 7.2: Underlying machine details

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

Table 7.3: Elapsed time overhead (in milliseconds) for provenance collection.

we tested Mule ESB independently with various communication protocols to send and receive data between different components of applications.

In this case, we calculated the time required for provenance collection, parsing and logging to *text* file. Different services which are provided as samples with Mule framework such as *Echo*, *Hello* and *WebApp* were tested. These samples use different configuration, i.e., communication protocols and data formats to send and receive data between various components. For example, *Echo* service is provided with two different configurations. The first configuration utilizes the *System.in* and *System.out* methods to send and receive data on console. The second configuration utilizes HTTP protocol to send and receive information. Where the *WebApp* service utilizes REST and SOAP protocols of communication to send and receive data in *XML* format.

We executed and tested *Echo*, *Hello* and *WebApp* services with different configuration and communication mechanisms multiple times. We calculate the *best time*, *worst time* and *average time* of the computation of provenance data. The process of testing is performed considering the overall provenance which includes the *MessageIn* and *MessageOut* phases of Mule. We also calculated provenance overhead only for *MessageIn* phase and provenance only for *MessageOut* phase. The underlying architecture and system details for this test case are presented in Table 7.4.

Figure 7.1 presents the performance results of various executions of the services with provenance (MessageIn), provenance (MessageOut) and provenance (MessageIn + MessageOut). The Y-axis represents the time required for collecting, parsing and storing of provenance data in *text* file.

The increase in time, i.e., the cost of provenance computation is calculated for overall provenance, only *MessageIn* provenance and only *MessageOut* provenance.

Component tested	Operating system	Services tested	Protocols tested	CPU architecture
Mule framework 2.1.2	CentOS 6.4	Echo	HTTP	x86_64 Intel (R) Core (TM)2
		WebApp	REST	
		Hello	SOAP	
			Console(Std-in/out)	

Table 7.4: Underlying architectural components used to evaluate Mule framework.

The cost is calculated for average values from multiple executions using Formula 7.1.

$$Time\,Increase = Averageof T_2 - Averageof T_1 \tag{7.1}$$

Where T_2 is the time including provenance and T_1 is the time excluding provenance.

The average increase in time for collecting, parsing and storing overall provenance data is only 1.12 milliseconds using Formula 7.1. The average increase in time for only *MessageIn* provenance is 0.2 milliseconds and for only *MessageOut* provenance is 1 millisecond when the comparison is performed for provenance collection with and without provenance data. The individual *MessageIn* and *MessageOut* provenance are collected and evaluated to observe the respective overhead in each phase. In a real e-Science environment, the overall provenance of any process is essential.

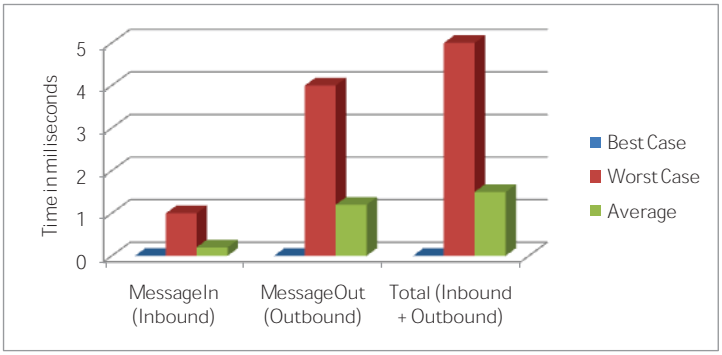


Figure 7.1: Evaluation of Mule ESB

7.1.3 Storage Overhead of the Framework

To find the storage overhead, we calculated the size of the XML file which is storing provenance data for CC and NC services. We chose a worst case scenario where all the incoming (Inflow) and outgoing (Outflow) data was stored, i.e., provenance

data without any filtering or parsing. This process was performed for every method of the Eucalyptus CC and NC services. The size of provenance data for some of the methods is represented in Figure 7.2 for Eucalyptus NC service. As shown in Figure 7.2 the average data size for the presented methods is about 2.24 Kilo Bytes (KB). There could be slight variation in the overall file size because the provenance data depends on the context of any request such as the methods that are invoked for a particular request in a Cloud IaaS. The rest of the methods in Eucalyptus CC and NC services demonstrated the same storage cost.

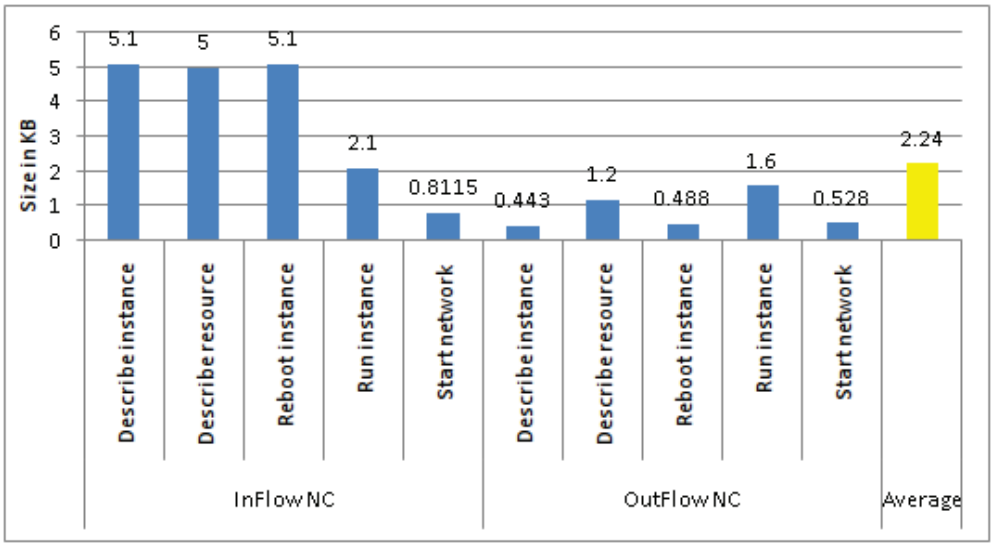


Figure 7.2: Various methods of Eucalyptus NC service and their respective storage overhead (kilobytes of disk space) for provenance data.

7.1.4 Discussion of the Results

The cost of collection of provenance data for IaaS module using our technique of interception is almost negligible as shown in Table 7.3 and Figure 7.1. Similarly the cost of storage of provenance data is also minimal as shown in Figure 7.2. It is essential to note that the low computation and storage overhead of the provenance frameworks is because of two reasons. Firstly, we used an approach where the extension of the middleware is achieved by utilizing built in feature, i.e., interceptors. This approach does not add any extra burden except the collection of provenance data. Therefore, the computation overhead is very low as provenance collection is part of the underlying architecture. Secondly, we utilized the link based mechanism to store the provenance data. This approach saves the space and time required to make a copy of the original object. Since the original objects already exist in

Cloud storage such as Walrus, therefore we do not make a copy of existing objects (contents) in Clouds.

In Eucalyptus or any other IaaS Cloud, a particular request maybe routed to various services and components. For example, to stream data in/out in Eucalyptus Cloud involves CLC and Storage Controller to interact and communicate for the successful operation. Similarly, requesting a VM involves CLC, CC and NC services to route a particular request. Therefore, the total overhead of provenance in IaaS Cloud is the summation of individual overheads of various services such as CLC, CC, and NC as depicted in Formula 7.2.

$$Total\,Overhead = \sum_{i=0}^{n} CLCi + \sum_{i=0}^{n} CCi + \sum_{i=0}^{n} NCi \quad (7.2)$$

Where the variable i represent the number of components such as methods for which the provenance module is configured. Therefore, the overall overhead depends on the configuration of provenance based on users requirements of granularity of data. The term CLC stands for Cloud Controller, CC for Cluster Controller, and NC for Node Controller. Figure 7.3 presents the total overhead where the individual values (average) of CLC, CC and NC are taken from Figure 7.1 and Table 7.3.

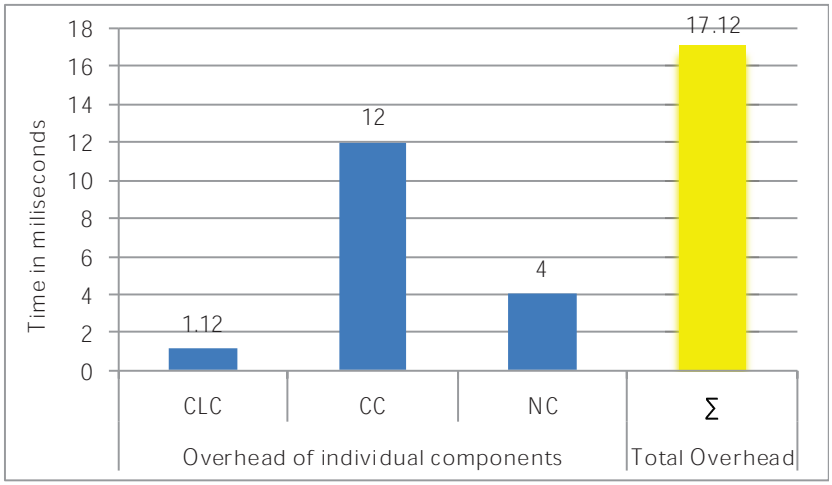


Figure 7.3: Total overhead (in milliseconds) of provenance computation is the summation of individual overheads from various components

7.2 Evaluation of the Framework for Platform and Software Models (Using WSO2 and DataSync Application)

The developed framework is tested and evaluated for platform model using WSO2 platform and DataSync application. The DataSync application which acts as a software model is deployed in WSO2. Two of the key servers in WSO2 platform, i.e., *Application Server* and *Enterprise Server Bus (ESB)* are tested with the web services *synctodatabase* and *syncfromdatabase* created in the DataSync application. Rest of the servers in WSO2, e.g., *Business Process Server*, *Identity Server* and *Data Services Server* etc. are tested with the default *Echo* service.

The *Application Server* in WSO2 is used for deploying applications such as business processes, workflows, web, and mobile applications. The *Application Server* utilizes open source technologies such as Apache Tomcat, Apache Axis2, and JAX-RS etc. to share various applications and business logic across an entire IT System. Similarly, *ESBs* utilize routing, meditation and transformation of messages to connect various applications or components of a particular application seamlessly in one place. For example, the format in XML, HTML and text are transformed and routed via REST and SOAP protocols among others. There are various transport protocols available to communicate with ESB such as HTTP, SMTP and HTTPs etc.

We performed the evaluation by using HTTP transport protocol, SOAP communication protocol, Axis2 (JAVA version) framework and XML data format for the communication and linking between different components of WSO2 framework and DataSync application. Table 7.3 presents the cost of provenance collection for different layers such as software and platform. The platform layer is evaluated with *Application Server* and *ESB* as shown in Table 7.3. The maximum times are the exceptional cases and therefore average time is calculated from multiple runs (i.e., 50 executions) of the involved components and layers. The average time presents the overhead for collection, parsing and storing of the provenance data.

Cloud Platform layer	Max time (ms)	Min time (ms)	Avg time (ms)
Software (DataSync application)	18	2	7
Platform (WSO2 AS)	22	1	3
Platform (WSO2 ESB)	12	1	2.5

Table 7.5: Elapsed time overhead (in milliseconds) of provenance collection for the platform and software layers

7.2.1 Discussion of the Results

The cost of collection of provenance for software and platform layers of Cloud is almost negligible as shown in Table 7.5. It is important to note that the cost of collection of provenance data is always in milliseconds for any layer of Cloud as depicted in Table 7.3, Table 7.5 and Figure 7.1 regardless of the underlying architectures. Hereby, the technique for provenance collection works in a standard fashion across all the layers of Cloud computing. Similarly the cost of storage of provenance data is also low as shown in Figure 7.2. The storage technique follows the *link based mechanism*, therefore the original objects size and their formats do not affect the size of provenance storage.

A particular application in Clouds may utilize one or more service models. For example, the DataSync application can be deployed directly in Eucalyptus Cloud without utilizing the platform. Similarly, the DataSync application can be deployed using the platform services. Therefore, the overall overhead of provenance depends on the architecture of applications. A generalized Formula 7.3 is presented which is used to calculate the grand total overhead with the summation of individual overheads from the software, platform and infrastructure layers.

$$Total\,Overhead = \sum_{i=0}^{n} \mathbf{SaaS}i + \sum_{i=0}^{n} \mathbf{PaaS}i + \sum_{i=0}^{n} \mathbf{IaaS}i \tag{7.3}$$

Where the variable i represent the number of components such as methods for which provenance module is configured in a particular service model. Figure 7.4 presents the total overhead where the values (average) of individual layers, i.e., IaaS, PaaS (Application Server) and SaaS (DataSync) are taken from Figure 7.3 and Table 7.5.

Depending on the granularity and storage mechanism, time required for provenance may slightly vary. The cost effective performance in terms of provenance computation and storage explains the utility of our provenance collection technique that follows the interceptor based approach and provenance storage that utilizes a link based approach. Given the overall advantages of provenance enabled Clouds such as fault tracking, resource utilization, patterns finding, trust, reliability and energy consumption, the extra overhead is negligible. Furthermore, the successful deployment of provenance collection to Axis2/C, Axis2 and Mule frameworks support our theory of a generalized and independent provenance framework.

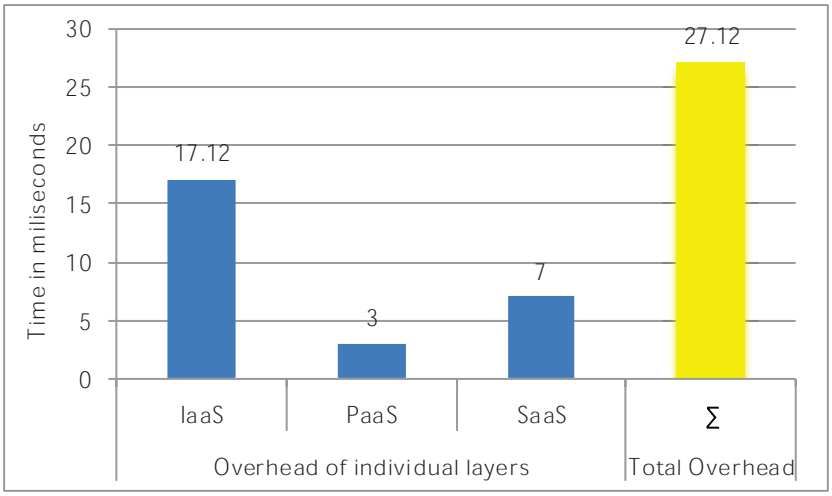


Figure 7.4: Total overhead (in milliseconds) of provenance computation is the summation of individual overheads from various layers

7.3 Evaluation of the Framework for Object Storage (STaaS Model)

We chose a subset of provenance data, i.e., metadata to evaluate the overhead involved in collecting, storing and querying provenance data for STaaS model. We performed various test cases as following:

- Collection Overhead: The first test case involves the overhead or cost of collecting and parsing the metadata by measuring the elapsed time for the server module, i.e., Walrus in Eucalyptus IaaS. The elapsed time is measured with and without the metadata, i.e., the modified and the original architecture of Eucalyptus Walrus.

- Storage Overhead: The second test case involves the storage overheads of the metadata in *XML* format and *NoSQL* schema.

- Query Performance: The third test case is performed to evaluate the query performance based on different protocols, i.e., *LinQ to XML* and SimpleDB APIs. The performance of the query protocols are compared with the native *hit and trial* method.

It is important to note that the overhead involved in metadata collection does not depend on the file size or format because the interception works in a seamless fashion regardless of the size and format of files. Similarly, the metadata storage also use link based mechanism and therefore the size and format of files are irrelevant. Therefore,

we expect the overhead or cost to be minimal in both cases, i.e., storage space and elapsed time using the interception techniques and link based mechanism.

7.3.1 Collection Overhead

To evaluate the collection overhead, we executed a scenario where various objects of different size and formats are uploaded to a Cloud with and without the metadata support. We steadily increased the number of objects from 200 to 5000 and calculated the time required to upload each additional 200 objects. The calculated overhead is measured via elapsed time between the objects with the metadata and without the metadata, i.e.,

Elapsed Time = Time taken by objects with the metadata minus the time taken by objects without the metadta

The overhead is displayed in Figure 7.5 and Figure 7.6. The results are calculated using a client/server architecture of three machines with various components of Eucalyptus Cloud installed on individual machines and the detail is provided in Table 7.6.

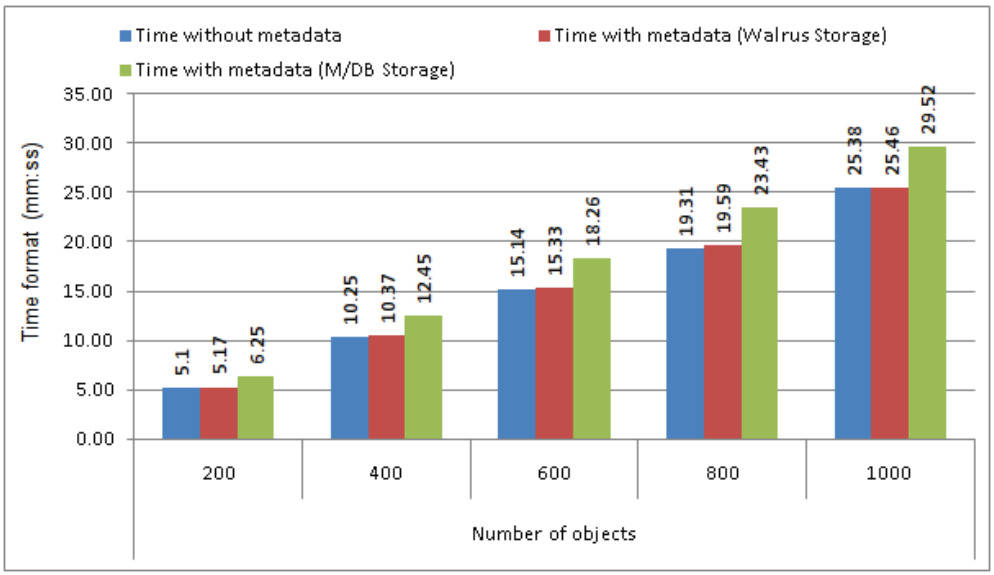


Figure 7.5: Results of the calculated time (minutes:seconds format) with and without the metadata for Eucalyptus Walrus

Elapsed time is calculated for different number of objects in Figure 7.5 and Figure 7.6. For instance, when the numbers of objects are 200, elapsed time with *XML* storage is **16 seconds** and with M/DB is **24 seconds**. Therefore the cost for individual objects is calculated using the equation 7.4.

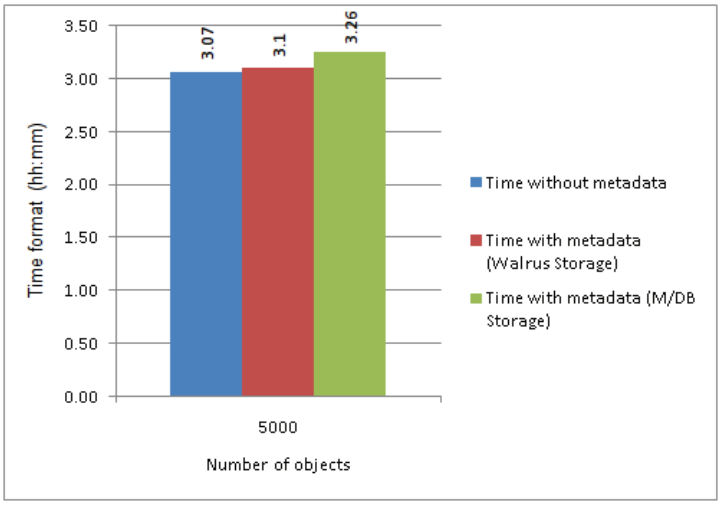


Figure 7.6: Results of the calculated time (hours:minutes format) with and without the metadata for Eucalyptus Walrus and 5000 objects

Resource	Operating System	Memory (MB)	Component Installed	Disk Size (GB)	CPU Architecture	CPU Cores	Network (Mb/s)
Machine 1 (Server)	Ubuntu 10.04	2048	CLC, Walrus	80	x84_64 Intel(R) Core (TM) 2	2 (2.33 GHz)	100
Machine 2 (Server)	CentOS 6.4	4192	Cluster, Node	250	x84_64 Intel(R) Core (TM)2	4 (2.83 GHz)	100
Machine 3 (Client)	Ubuntu 12.04	2048	Amazon SDK, JetS3t	80	x84_64 Intel(R) Core (TM) 2	2 (2.13 GHz)	100
Virtual Machine Instance	Ubuntu 10.04	512	M/DB	5 GB	x84_64 Intel(R) Core	1 (2.83 GHz)	100

Table 7.6: System details for evaluation

$$Cost = \frac{ETi}{i} \tag{7.4}$$

where i represent the number of objects and ET represents the elapsed time.

Figure 7.7 presents the elapsed time for different number of objects using Formula 7.4. The results in Figure 7.7 clearly show that the involved overhead is negligible when individual objects are uploaded to Cloud. Moreover, it also demonstrates, the cost for individual object remains almost the same regardless the number of objects as depicted by the trend lines. Furthermore, object storage is reflecting

a slightly better performance when compared to M/DB storage of metadata. We believe the reason is the configuration of M/DB in the Cloud on a virtual machine instance that is running on a different physical machine (Node Controller). Each time the server component request to add the metadata in M/DB, network and HTTP protocol is involved. This overhead can be decreased if metadata is stored in memory for a particular number of objects and then the whole bunch is written to the M/DB. By using this approach, the number of request over HTTP protocol will be reduced and therefore the overall overhead can also be reduced. This approach can create consistency problem between the original data and metadata. For instance, when the data is stored in memory for particular number of objects, meanwhile another user can modify the existing data. In general, the overhead increases as the number of objects increases but remains minimal.

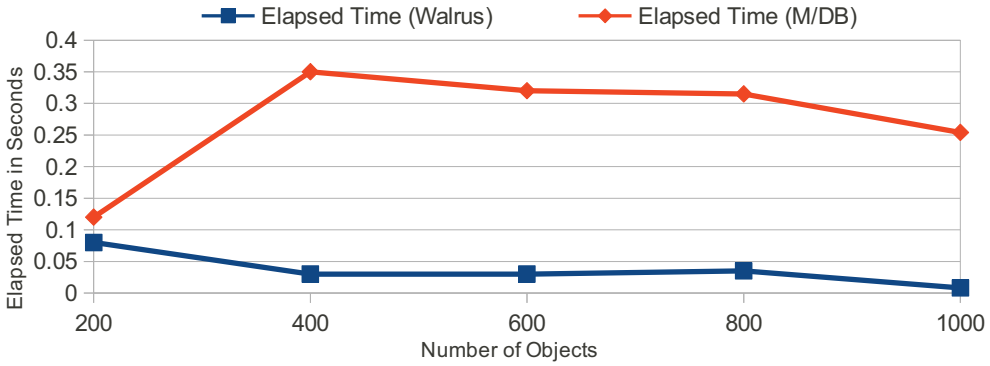


Figure 7.7: Cost of provenance collection in terms of elapsed time (in seconds) for different number of objects

7.3.2 Storage Overhead

When metadata is stored in *XML* object, each individual metadata item cost approximately *1.8 KB* of disk space which includes user metadata, system metadata and server metadata. On the other hand, when M/DB is utilized to store the metadata, Formula 7.5 is used to measure the base storage required for various items and their respective attributes and values pairs. In the given Formula, storage size varies

depending on the text length of item names, attribute names and attribute values.

$$Storage\ Overhead = \sum_{i=1}^{n} (ItemNamesSizeBytes)_i + \sum_{i=1}^{n} (AttributeNamesSizeBytes)_i + \sum_{i=1}^{n} (AttributeValuesSizeBytes)_i \quad (7.5)$$

Where the variable i represent the number of objects that are uploaded to a Cloud storage. Figure 7.8 presents the cost of provenance storage using the average size of 1.8 KB of disk space for XML repository, and the Formula 7.5 for M/DB repository. Figure 7.8 depicts a marginal cost of provenance storage which remains consistent regardless the size of the original objects uploaded to the Cloud. This is achieved using the link based approach and coarse grained provenance for collection and storage.

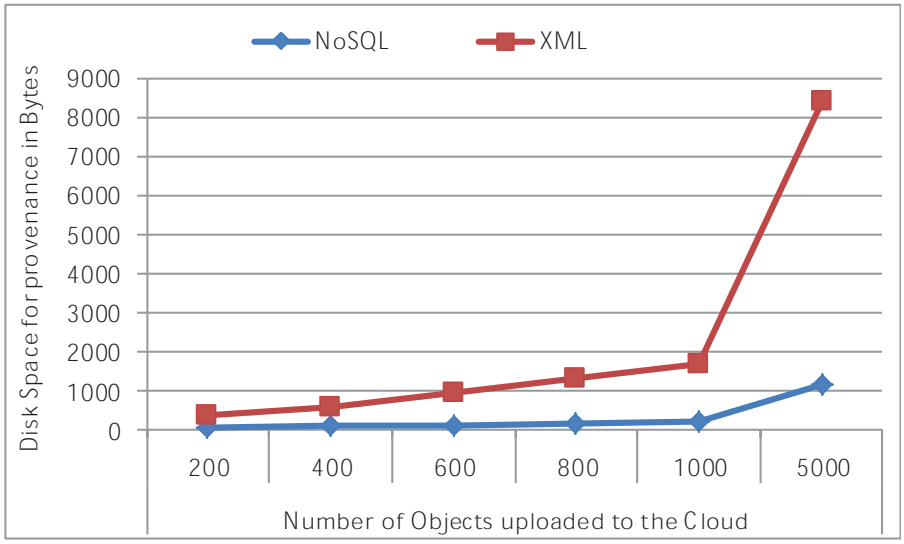


Figure 7.8: Cost of provenance storage (disk space in bytes) for different number of objects

7.3.3 Query Performance

The performance of query protocols for the stored metadata in different formats is evaluated in Section 5.1.7.

7.3.4 Discussion of the Results

The framework presents a marginal cost in terms of provenance collection and storage overheads as shown by various experiments for STaaS model just like the IaaS, PaaS, and SaaS models of the Cloud. For the storage model, it is also proved that the framework technique is not affected by the size of objects in Clouds. Therefore, the interception technique and *link based mechanism* for provenance collection and storage proved their utility. Moreover, the results from Section 5.1.7 clearly depicts the efficiency of query protocols utilized by the framework. Hence, the framework presents a consistent evaluation with a marginal cost for all the layers or types of Clouds paradigm.

7.4 Extension of the Provenance Framework to Nimbus Cloud

The Nimbus project [101] is an open source Cloud IaaS model which provides two products named the *Nimbus Infrastructure* and the *Nimbus Platform*.

The *Nimbus Infrastructure* allows clients to acquire various resources, deploy virtual machines on the acquired resources and configure them according to the user or application requirements. This part of Nimbus Cloud also provides the implementation of a storage Cloud which is called *Cumulus* [33]. The communication between a client, *Nimbus Infrastructure* and *Cumulus* is established using WSRF, WSDL and REST mechanisms. Furthermore, *Nimbus Infrastructure* is compatible with Amazon EC2 and *Cumulus* is compatible with Amazon S3 services. Figure 7.9 depicts the main components of *Nimbus Infrastructure* where the description of the components is following:

- Cloud Client: The client provides various APIs that are utilized by users and applications to communicate with the infrastructure services.
- Service Node: The component which sits in the middle between the client and the VM. The job of the Service Node is to manage various requests from users such as creation of a Virtual Machine and/or storing data. The storage component stores users data along with raw Virtual Machines, i.e., resources.
- Virtual Machine Manager (VMM): The component in Nimbus Cloud which handles the management of virtual machines. It uses libvirt library for the management of XEN/KVM Virtual Machines and DHCP server is utilized to assign IP addresses (public and private) to the Virtual Machines.

The *Nimbus Platform* provides additional tools to simplify the management of the infrastructure services such as the integration of *Nimbus Infrastructure* with other Cloud IaaS models, e.g., OpenStack and Amazon. To test the developed provenance framework, we chose the *Nimbus Infrastructure*.

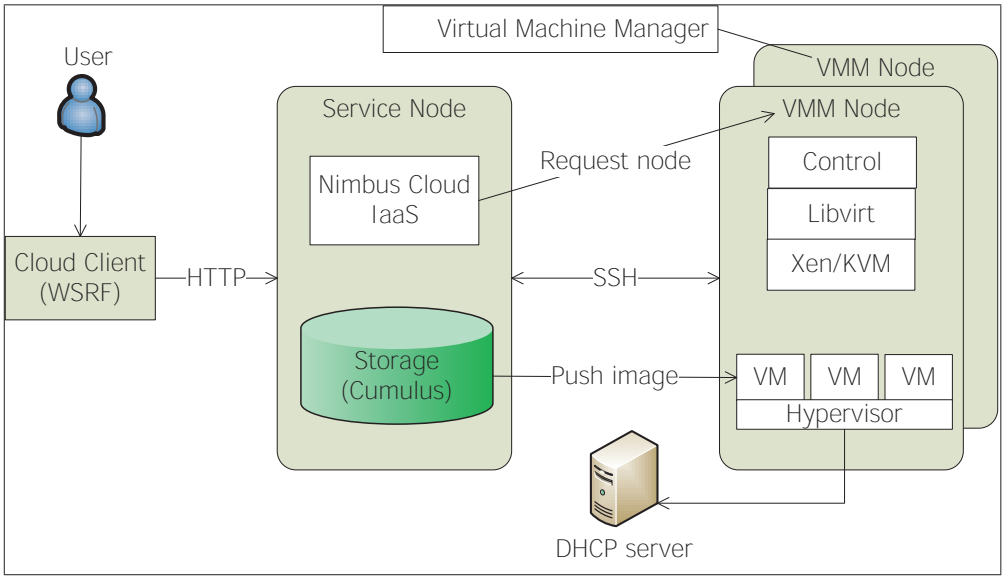


Figure 7.9: Various components in the Nimbus IaaS Cloud.

Any request from a user is routed through the components presented in Figure 7.9. To collect the provenance data in the Nimbus Cloud, the *Service Node* resides between the *Cloud Client* and VMM. Various services in the *Service Node* are exposed via WSDL. These services utilize the Apache Axis2 framework for the communication and linking mechanisms.

The developed provenance framework in divided into various components of collection, storage, query and visualization. These components work independent from each other, i.e., the modularity of the framework. Therefore, to test the integration of our provenance framework, the collection module 4.2.1 needs to be tested. We deployed the provenance collection module from the developed framework in Nimbus Cloud at the *Service Node*. It is to be noted that the same module is used which is deployed in WSO2 platform because they both use the same Apache Axis2 framework. The test was successful and we were able to engage the provenance module and collect various provenance data as shown in Listing 7.1.

Listing 7.1 provides a sample of the provenance data collected from Nimbus IaaS Cloud where various parameters are presented such as the location of Nimbus services, type of a request, networking mode, CPU architecture, memory, CPU cores, and the state of the resource among others.

Listing 7.1: Sample of provenance data collected from the Nimbus IaaS Cloud

```
<?xml version="1.0" encoding="UTF-8"?>
<To>https://alf-laptop:8443/wsrf/services/WorkspaceFactoryService</To>
    !- the address or machine where the request is forwarded-
<Action>http://www.globus.org/2008/06/workspace/
   WorkspaceFactoryPortType/createRequest</Action> !-the type of
   request which is the creation of a virtual machine in this sample-
<networking> !- the internel networking interface of Nimbus Cloud-
  <name>eth0</name>
    <ipConfig>
     <acquisitionMethod> AllocateAndConfigure </acquisitionMethod>
    </ipConfig>
  <association> public </association> !- the public address assigned
     to the resource and utilized by users-
</networking>
<CPUArchitecture> !- the architecture of the assigned Virtual machine,
    i.e., CPU-
  <CPUArchitectureName> x86 </CPUArchitectureName>
</CPUArchitecture>
<VMM>
  <type>Xen</type> !- the vitrualization technique-
  <version>3</version>
</VMM>
<diskCollection> !- the resource which is requested along with its
   location-
  <location>cumulus://alf-laptop:8888/Repo/VMS/35df3a8c-afd7-11e1-a77d
     -0015c553233a/ubuntu10.10 </location>
</diskCollection>
<mountAs>hda</mountAs> !- the type of mounting-
<permissions>ReadWrite</permissions> !- the permissions assigned to
   the resource-
<DeploymentTime>
  <minDuration>PT60M</minDuration> !- the time duration for which the
     resource is requested-
</DeploymentTime>
<InitialState>Running</InitialState> !- state of the virtual machine-
<IndividualCPUCount>
  <Exact>1.0</Exact> !- number of assigned CPU-
</IndividualCPUCount>
<IndividualPhysicalMemory>
  <Exact>256.0</Exact> !- assigned memory in MB-
</IndividualPhysicalMemory>
<NodeNumber>1</NodeNumber> !-each instance is assigned a unique number
   -
<ShutdownMechanism>Trash</ShutdownMechanism> !- The operation to be
   perfomrmed when the machine is shut down-
```

7.5 Conclusion

In this chapter we focused on two key points, i.e., to measure the cost of the developed framework using the overhead involved and to evaluate the independence of the framework from underlying architectures and domains.

Firstly, we evaluated the cost of computation, storage and query for various modules of the framework. We performed tests to evaluate the overhead involved in collecting and storing provenance data using various transport protocols, communication protocols and data formats. Furthermore, we focused on various service models, i.e., types of Cloud computing and the respective cost of various modules of the framework. As shown with different experiments that the related overhead of computation and query is always in milliseconds which we consider to be minimal. The storage overhead depends on the granularity of provenance data and the technique to store provenance data. We calculated the storage overhead involved in different service models of Cloud and detailed the results. Since we are using the link based mechanism to store the provenance data, therefore the storage overhead is also low. We used different approaches to store the provenance data, i.e., *XML* format in object storage and NoSQL schema for M/DB. We believe that the overhead involved in collecting and storing provenance data is negligible when considering the utility of provenance in Clouds such as energy consumption, fault tracking, content based searching, trust, reliability, pattern recognition and behavior analysis among others.

Secondly, we provided a proof of concept with examples that the proposed technique of interception for provenance data can be deployed across different infrastructure, platform, and software layers of Clouds. The successful deployment of the framework across various layers of Cloud and with a different IaaS Cloud, i.e., Nimbus proved that our approach is independent from underlying architectures and domains.

8. Conclusion

Provenance is associated with datasets, processes, applications, infrastructures and platforms that are used to derive a final result or data product in computational or data science. Provenance provides useful information to understand, reproduce and/or validate the final result along with the creator process and sub processes. With the recent trend in distributed computing, Clouds are becoming the future platform for e-Science research such as private Clouds which are used in lab environments and universities. The distributed and abstract architecture of Clouds has attracted the research and business communities for applications deployment such as storage and sharing of data, and computation of data and processes.

The dynamic architecture of Clouds provides different service models such as IaaS, PaaS, SaaS, and various characteristics such as abstraction, scalability, on-demand computing among others. The different service models or layers address various view points such as consumer, provider and developer of the Cloud. The existing research of provenance in distributed computing like grids and workflows is mostly focused on the application tier. With Clouds, the provenance of the architecture, i.e., service models or layers is also essential, e.g., to explore the connections between the layers.

It is extremely important to provide provenance of the dynamic architecture of Clouds from the perspectives of various service models and view points. Introducing provenance in Clouds result in the reliability and trust of data and processes along with other potential applications such as content based searching, fault tracking of Cloud services (e.g., IaaS services), debug applications and utilization of resources etc. Moreover, provenance of Clouds helps to understand the relationships that exist between various layers or types of Clouds.

In this thesis, we investigated two key questions, i.e., *why* and *how* to provide provenance of Clouds. The *why* part is focused on the implication of provenance enabled Clouds, i.e., the usefulness for the end users when provenance is augmented in the Cloud. This part is build up with motivation, problem definition and the various applications based on provenance data. The applications are explored from the view points of consumers, developers and resource providers with respective service mod-

els of Clouds. The *how* part of this thesis provides the design and implementation of a modular and independent framework for the management (collection, parsing, storage, query, and visualization) of provenance data. The developed framework is uniformly applied to the underlying abstract and distributed architecture of Clouds such as different service models, i.e., *infrastructure*, *platform*, *storage* and *software* as the proof of concept. Hereby, the evaluation of the framework is performed with respect to various application domains and the framework components themselves.

8.1 Research Contributions

This thesis investigates the significance of provenance of Clouds which is motivated by the recent shift of computation and storage towards Cloud. To satisfy the *research questions* and the overall goal, i.e., *Provenance in Clouds: Framework, Applications and Implication*, this thesis is divided into multiple parts.

The first part introduces provenance with respect to the architecture of Cloud computing and provides the reasoning to incorporate provenance in Clouds. Hereby, the architecture of Clouds is explored from the view point of a user, developer and resources provider along with the respective provenance data. A list of requirements/challenges are presented for the collection of provenance based on the distributed, abstract, scalable and layered architecture of Clouds. In addition, requirements such as low cost of provenance collection and storage are also presented.

The second part of this thesis provides the design of the framework for the collection and management of provenance which addresses various requirements and challenges offered by the layered and abstract architecture of Cloud computing. The framework adopts a modular and independent approach for the collection of provenance. The framework also provides services such as provenance storage, query, and visualization. The implementation of the framework for different service models of Cloud is also provided.

The third part details the implication of provenance in Clouds by defining, implementing and validating various applications based on the collected provenance and the components of the framework itself. These applications are explored from the view points of a consumer, developer and resources provider in Cloud. Moreover, the aggregated provenance is provided which exploits the relationships between the layers such as finding the root cause of any failure. Lastly, the framework is tested and evaluated for different service models of Cloud.

Figure 8.1 depicts the various components and activities which are detailed in this thesis. To support the provenance enabled Clouds and satisfy the research questions, this thesis provides the followings:

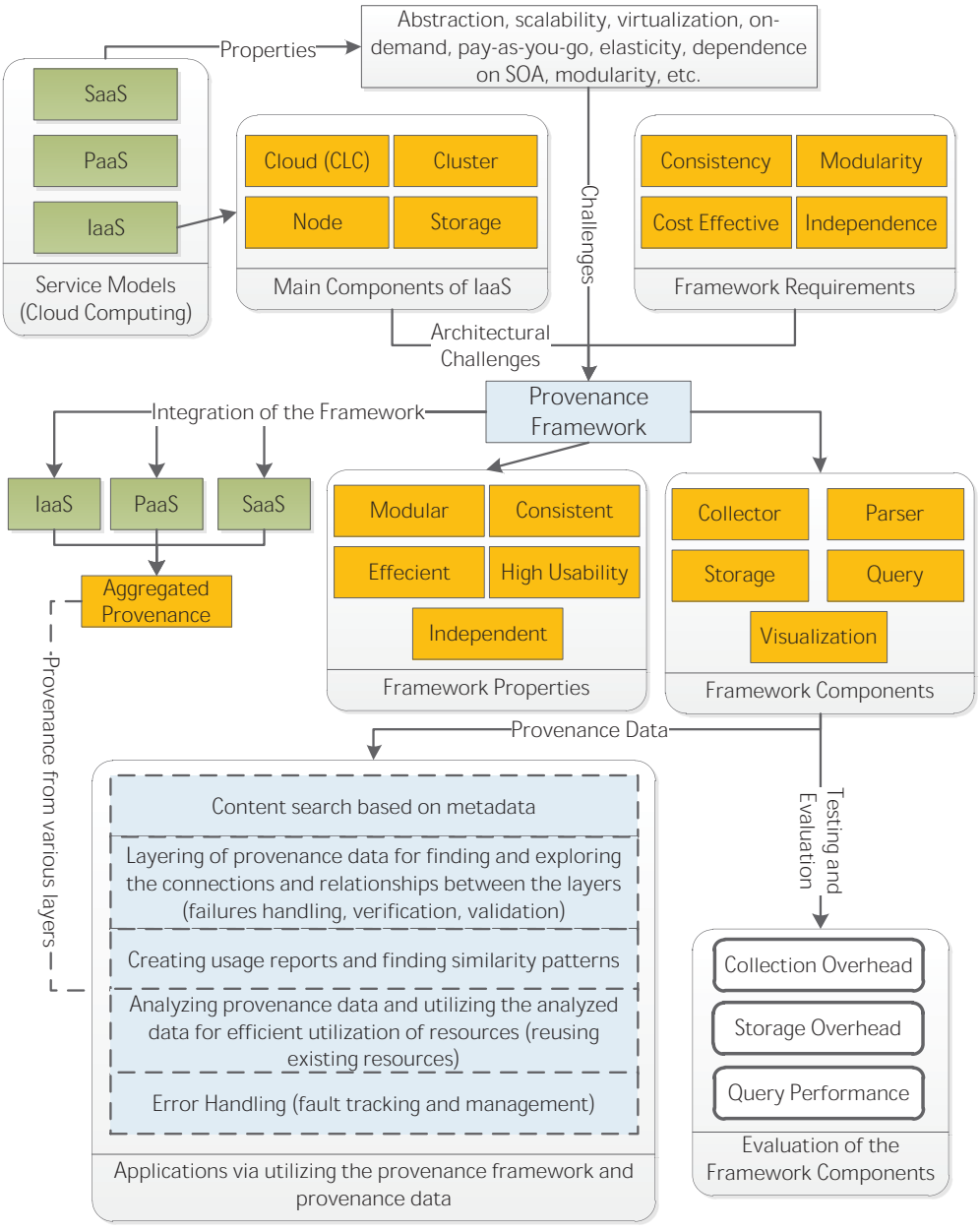


Figure 8.1: Major Contributions of Thesis in Pictorial Form

- **Cloud Computing:** We studied the dynamic architecture of Cloud computing such as different service models of IaaS, PaaS, SaaS and STaaS (object storage), and the properties of Clouds such as abstraction, scalability and on-demand computing etc. We provided examples with each service model, i.e., Eucalyptus, WSO2, DataSync and Walrus. We gave a detailed view of the design and characteristics of Cloud computing. A list of requirements/challenges is provided based on the design and characteristics that must be addressed while providing provenance in Clouds. Moreover, a list of significant provenance items along with their description is provided for the individual layers of Cloud.

- **Analyzing Existing Techniques of Provenance:** We provided a state of the art for the collection of provenance data from existing fields such as grid, workflow and Cloud computing. We categorized the existing schemes into two main types (provenance as core part of the Cloud and provenance as an independent service) and presented their impact on Cloud computing. This involved various advantages and disadvantages of the existing schemes assuming they are incorporated in Clouds. We presented in detail the pros and cons when provenance collection is part of the Cloud core (the underlying architecture or service models) and when provenance collection is independent of the Cloud core.

- **Proposed Provenance Framework:** We designed and developed a provenance framework for Clouds which: (i) addresses the abstract, modular and distributed architecture, (ii) addresses the list of requirements for the collection of provenance data in Cloud computing, and (iii) takes the advantages from the existing schemes of provenance from workflow and grid computing. The framework adopts a modular design and independent approach for the collection and management of provenance data. The framework allows incorporating provenance in Clouds with minimal knowledge and understanding of the underlying architectures and domains. Furthermore, the framework is extended towards different service models with minimal changes. The developed framework supports properties such as modularity, independence from underlying architectures, marginal cost of provenance collection and storage, and high usability among others. The implementation of the framework and its various components is also provided using various types/layers of Clouds.

- **Applications of Provenance in Clouds:** The collection and management of provenance is one aspect in any domain of e-Science. The other aspect is to utilize such data, e.g., in Clouds to make them useful for the end users.

Therefore, we presented and validated various applications exploiting the collected provenance and the components of the framework from the view point of consumers, developers and resource providers. The applications are designed and tested to cover various aspects of provenance enabled Clouds such as: (i) metadata based search in Clouds object storage, (ii) report generation of the usage of Cloud resources and, finding behavior and similarity patterns in the usage of Cloud resources, and (iii) resource utilization by analyzing and using provenance data.

- **Aggregated Provenance:** The granularity of provenance data depends on the architecture of Cloud computing. Since, Clouds are composed of different layers such as IaaS, PaaS, SaaS and STaaS among others, provenance data can be divided into different categories of *infrastructure*, *platform*, *software*, *virtual machines*, *web browsers* and *physical machines* etc. Similarly, provenance data inside one layer can be further divided into different components such as provenance of CLC, CC and NC of the Eucalyptus Cloud (IaaS model). The aggregated provenance from the different layers of Clouds is extremely important to: (i) understand and explore the complete architecture, (ii) find the relationships between layers, (iii) explore the interaction between layers, and (iv) answer various queries which require individual or the aggregated provenance from multiple layers.

  A DataSync application was developed and explored for the layered architecture of the Cloud with respective view points of consumers, developers and resource providers. Using the DataSync application and the layered architecture of Clouds, aggregated provenance was established. The aggregated provenance was further exploited for finding the root cause of various failures, i.e., fault tracking. This was accomplished utilizing the relationships between the layers of Clouds and the users activities (using DataSync) in aggregated provenance. Moreover, the aggregated provenance was used to find components such as data and processes which are affected because of the failure.

- **Evaluation:** Any layer of integration to solve a problem such as provenance requires extra computation and storage overhead. We provided various evaluation results that our framework requires minimal to negligible overhead of the computation and storage of the provenance data. Furthermore, we proved that the framework is extensible to different service models, i.e., PaaS and SaaS and other Clouds such as Nimbus. We also provided the performance of query protocols which utilize provenance data from *XML* schema and NoSQL database. The results clearly showed that the design of the framework using

interception for the collection and *link based mechanism* for the storage has marginal cost and provides fast and efficient query mechanisms.

8.2 Limitations, Open Issues and Future Work

Some of the limitations, issues and future work directions for provenance in Clouds are following:

- **A completely independent provenance scheme:** Our framework relies on the extension of underlying architectures, i.e., middlewares. On one hand, this approach makes our framework independent of various domains and applications. On the other hand, the framework becomes dependent on the middlewares. Due to the extension of middlewares, the proposed framework can collect the data which is passing through various components and services via middlewares. If an organization requires fine grained provenance such as the data which is not passing through various components of middlewares, the framework is not able to collect such data. It would be interesting to investigate a scheme which can communicate with components and services of Clouds from outside. The major challenge to provide such a scheme is the inability to extend and communicate with business Clouds and services. Furthermore, it would be interesting to layer the provenance data where all the service models of Cloud are utilized at the same time. For example, the support of WSO2 platform is provided in the commercial version of Eucalyptus. It would be compelling to layer the provenance data of an application that is developed and deployed using a platform which utilizes resources from infrastructure in commercial offerings.

- **Security and privacy issues:** The security and privacy of data and processes are concerns of any organization, especially in the context of sensitive and critical private data. We do not provide any direct mechanism or algorithm to handle such issues. However, we provide indirect methods to address such issues, e.g., a provider can choose to disable the collection and storage of provenance for the clusters and nodes where sensitive data is stored and critical processes are executed. The same technique can be applied to disable the provenance collection from the platform and software layers for various components and services.

- **Future work:** The future work can extend the framework to its full potential such as the implementation of the framework for business Clouds. Similarly,

the service models can be investigated and probably extended by using a completely independent approach for provenance data. It will also be interesting to express provenance data using Web Ontology Language (OWL) and Resource Description Framework (RDF) technologies for semantic purposes. Furthermore, various potential applications can be proposed based on the provenance data and framework such as resolution (proactive or reactive approach) of faults and errors.

8.3 Summary

Provenance is an important ingredient in various domains of computation and data science for the purpose of verification, audit trials, reproduction, trust and reliability. With the shift of dynamics toward Clouds such as computation and storage, it is realized that Clouds themselves have important provenance from the *infrastructure*, *platform*, *storage* and *software* layers. However, the existing research is mostly focused in the provenance of application layer of distributed computing. To the best of our knowledge, provenance is still missing to its full potential in Cloud computing such as provenance of Clouds themselves. Therefore, we analyzed the dynamic and modular architecture of Cloud computing from different view points (consumers, developers and resource providers) and list the requirements for the collection and management of provenance data. Hereby, we extend the architecture of Clouds in a structured way to provide a framework for the collection and management of provenance data and detailed various applications which take advantage of the collected data and the developed framework.

The applications which are explored in this thesis address the view points of a user and resource provider in detail. The software developer point of view is also discussed while providing the aggregated provenance from various layers of Cloud using the DataSync application. Various applications which utilize the collected provenance data and developed framework provide the proof of concept, results and the practical implication of the significance of provenance in Clouds. For instance, providing the ability to search objects in Clouds explains the utilization of the framework and the collected provenance data.

The framework is designed and implemented with key points such as, (i) modular design, (ii) independent from underlying architectures and domains, (iii) extensible to various layers of Clouds with minimum changes, (iv) easy to use and users controlled, and (v) low cost of provenance computation and storage which is evident through various evaluations performed in this thesis.

Appendices

A.1 XML Configuration Files of Mule (Eucalyptus Cloud)

Various configuration files (XML files) which are used by the Mule framework are provided below. To configure, i.e., enable/disable provenance, these configuration files need to be edited for the corresponding flow, model, service and/or method accordingly.

A.1.1 eucalyptus-walrus

Listing: eucalyptus-walrus.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org/schema/mule/core/2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:spring="
      http://www.springframework.org/schema/beans"
  xmlns:vm="http://www.mulesource.org/schema/mule/vm/2.0" xmlns:euca="
      http://www.eucalyptus.com/schema/cloud/1.6"
  xsi:schemaLocation="
       http://www.springframework.org/schema/beans http://www.
           springframework.org/schema/beans/spring-beans-2.0.xsd
       http://www.mulesource.org/schema/mule/core/2.0 http://www.
           mulesource.org/schema/mule/core/2.0/mule.xsd
       http://www.mulesource.org/schema/mule/vm/2.0 http://www.
           mulesource.org/schema/mule/vm/2.0/mule-vm.xsd
       http://www.eucalyptus.com/schema/cloud/1.6 http://www.eucalyptus
           .com/schema/cloud/1.6/euca.xsd">
    <model name="walrus">
        <default-service-exception-strategy>
            <outbound-endpoint ref="WalrusReplyQueueWS" />
        </default-service-exception-strategy>
        <service name="WalrusRequestQueue">
            <inbound>
                <inbound-endpoint ref="WalrusRequestQueueEndpoint" />
            </inbound>
            <bridge-component/>
            <outbound>
              <filtering-router>
        <outbound-endpoint ref="BukkitWS" />
        <payload-type-filter
      expectedType="edu.ucsb.eucalyptus.msgs.WalrusComponentMessageType
          " />
    </filtering-router>
        <filtering-router>
```

```
                    <outbound-endpoint ref="BukkitWS" />
                    <payload-type-filter expectedType="edu.ucsb.
                        eucalyptus.msgs.WalrusRequestType" />
                </filtering-router>
            </outbound>
        </service>
        <service name="Bukkit">
            <inbound>
                <inbound-endpoint ref="BukkitWS" />
            </inbound>
            <component class="edu.ucsb.eucalyptus.cloud.ws.
                WalrusControl" />
            <outbound>
                <outbound-pass-through-router>
                    <outbound-endpoint ref="WalrusReplyQueueWS" />
                </outbound-pass-through-router>
            </outbound>
        </service>
        <service name="WalrusReplyQueue">
            <inbound>
                <inbound-endpoint ref="WalrusReplyQueueWS" />
            </inbound>
            <component class="com.eucalyptus.ws.util.ReplyQueue" />
        </service>
    </model>
    <model name="WalrusInternal">
        <service name="BukkitInternal">
            <inbound>
                <inbound-endpoint ref="BukkitInternalWS" />
                <inbound-endpoint ref="BukkitInternalVM" />
            </inbound>
            <component class="edu.ucsb.eucalyptus.cloud.ws.
                WalrusControl" />
        </service>
    </model>
</mule>
```

A.1.2 eucalyptus-verification

Listing: eucalyptus-verification.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org/schema/mule/core/2.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:vm="http://www.mulesource.org/schema/mule/vm/2.0"
      xmlns:euca="http://www.eucalyptus.com/schema/cloud/1.6"
      xsi:schemaLocation="
       http://www.springframework.org/schema/beans http://www.
           springframework.org/schema/beans/spring-beans-2.0.xsd
       http://www.mulesource.org/schema/mule/core/2.0 http://www.
           mulesource.org/schema/mule/core/2.0/mule.xsd
```

```
    http://www.mulesource.org/schema/mule/vm/2.0 http://www.
        mulesource.org/schema/mule/vm/2.0/mule-vm.xsd
    http://www.eucalyptus.com/schema/cloud/1.6 http://www.eucalyptus
        .com/schema/cloud/1.6/euca.xsd">
  <model name="eucalyptus-verification">
    <default-service-exception-strategy>
      <outbound-endpoint ref="ReplyQueueEndpoint" />
    </default-service-exception-strategy>
    <service name="StartVerify">
      <inbound>
        <inbound-endpoint ref="StartVerifyWS" />
      </inbound>
      <component class="com.eucalyptus.sla.VmAdmissionControl" />
      <outbound>
        <outbound-pass-through-router>
          <outbound-endpoint ref="ImageVerifyWS" />
        </outbound-pass-through-router>
      </outbound>
    </service>
    <service name="FinishedVerify">
      <inbound>
        <inbound-endpoint ref="FinishedVerifyWS" />
      </inbound>
      <component class="com.eucalyptus.sla.VmAdmissionControl" />
      <outbound>
        <outbound-pass-through-router>
          <outbound-endpoint ref="UpdateSystemWS" />
        </outbound-pass-through-router>
      </outbound>
    </service>
    <service name="ImageVerify">
      <inbound>
        <inbound-endpoint ref="ImageVerifyWS" />
      </inbound>
      <component class="com.eucalyptus.images.ImageManager" />
      <outbound>
        <outbound-pass-through-router>
          <outbound-endpoint ref="KeyPairVerifyWS" />
        </outbound-pass-through-router>
      </outbound>
    </service>
    <service name="KeyPairVerify">
      <inbound>
        <inbound-endpoint ref="KeyPairVerifyWS" />
      </inbound>
      <component class="com.eucalyptus.keys.KeyPairManager" />
      <outbound>
        <outbound-pass-through-router>
          <outbound-endpoint ref="VmTypeVerifyWS" />
        </outbound-pass-through-router>
      </outbound>
    </service>
    <service name="VmTypeVerify">
```

```
            <inbound>
                <inbound-endpoint ref="VmTypeVerifyWS"/>
            </inbound>
            <component class="com.eucalyptus.cluster.VmTypeVerify"/>
            <outbound>
                <outbound-pass-through-router>
                    <outbound-endpoint ref="GroupsVerifyWS"/>
                </outbound-pass-through-router>
            </outbound>
        </service>
        <service name="GroupsVerify">
            <inbound>
                <inbound-endpoint ref="GroupsVerifyWS"/>
            </inbound>
            <component class="com.eucalyptus.network.
                NetworkGroupManager"/>
            <outbound>
                <outbound-pass-through-router>
                    <outbound-endpoint ref="FinishedVerifyWS"/>
                </outbound-pass-through-router>
            </outbound>
        </service>
    </model>
</mule>
```

A.1.3 db-model

Listing: db-model.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org/schema/mule/core/2.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:vm="http://www.mulesource.org/schema/mule/vm/2.0"
      xmlns:euca="http://www.eucalyptus.com/schema/cloud/1.6"
      xsi:schemaLocation="
       http://www.springframework.org/schema/beans http://www.
           springframework.org/schema/beans/spring-beans-2.0.xsd
       http://www.mulesource.org/schema/mule/core/2.0 http://www.
           mulesource.org/schema/mule/core/2.0/mule.xsd
       http://www.mulesource.org/schema/mule/vm/2.0 http://www.
           mulesource.org/schema/mule/vm/2.0/mule-vm.xsd
       http://www.eucalyptus.com/schema/cloud/1.6 http://www.eucalyptus
           .com/schema/cloud/1.6/euca.xsd">
    <model name="DB">
    </model>
    <model name="DBInternal">
    </model>
</mule>
```

A.1.4 eucalyptus-bootstrap

Listing: eucalyptus-bootstrap.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org/schema/mule/core/2.0" xmlns:xsi=
    "http://www.w3.org/2001/XMLSchema-instance"
  xmlns:spring="http://www.springframework.org/schema/beans" xmlns:euca
      ="http://www.eucalyptus.com/schema/cloud/1.6"
  xmlns:vm="http://www.mulesource.org/schema/mule/vm/2.0" xmlns:mule="
      http://www.mulesource.org/schema/mule/core/2.0"
  xsi:schemaLocation="
       http://www.springframework.org/schema/beans http://www.
           springframework.org/schema/beans/spring-beans-2.0.xsd
       http://www.mulesource.org/schema/mule/core/2.0 http://www.
           mulesource.org/schema/mule/core/2.0/mule.xsd
       http://www.mulesource.org/schema/mule/vm/2.0 http://www.
           mulesource.org/schema/mule/vm/2.0/mule-vm.xsd
       http://www.eucalyptus.com/schema/cloud/1.6 http://www.eucalyptus
           .com/schema/cloud/1.6/euca.xsd">
  <mule:configuration>
    <mule:default-threading-profile doThreading="true" maxThreadsActive
        ="16" poolExhaustedAction="RUN" />
  </mule:configuration>
   <euca:connector name="eucaws" />
  <vm:connector name="internal-async" queueEvents="false" >
    <vm:queueProfile persistent="false" maxOutstandingMessages="256"/>
  </vm:connector>
  <endpoint name="ReplyQueueEndpoint" address="vm://ReplyQueue"
      synchronous="false" />
  <endpoint name="RequestQueueEndpoint" address="vm://RequestQueue"
      synchronous="false" />
  <endpoint name="EucalyptusRequestQueueEndpoint" address="vm://
      EucalyptusRequestQueue" synchronous="false" />
  <endpoint name="WalrusRequestQueueEndpoint" address="vm://
      WalrusRequestQueue" synchronous="false" />
  <endpoint name="StorageRequestQueueEndpoint" address="vm://
      StorageRequestQueue" synchronous="false" />
  <endpoint name="ComponentRequestQueueEndpoint" address="vm://
      ComponentRequestQueue" synchronous="false" />
  <model name="eucalyptus-bootstrap">
    <default-service-exception-strategy>
      <outbound-endpoint ref="ReplyQueueEndpoint" />
    </default-service-exception-strategy>
    <service name="ReplyQueue">
      <inbound>
        <inbound-endpoint ref="ReplyQueueEndpoint" />
      </inbound>
      <component class="com.eucalyptus.ws.util.ReplyQueue" />
    </service>
    <service name="RequestQueue">
      <inbound>
        <inbound-endpoint ref="RequestQueueEndpoint" />
```

```
      </inbound>
      <component class="com.eucalyptus.ws.util.RequestQueue" />
      <outbound>
        <filtering-router>
          <outbound-endpoint ref="WalrusRequestQueueEndpoint"
              synchronous="false" />
          <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
              WalrusRequestType" />
        </filtering-router>
        <filtering-router>
          <outbound-endpoint ref="StorageRequestQueueEndpoint"
              synchronous="false" />
          <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
              StorageRequestType" />
        </filtering-router>
        <filtering-router>
          <outbound-endpoint ref="EucalyptusRequestQueueEndpoint"
              synchronous="false" />
          <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
              EucalyptusMessage" />
        </filtering-router>
        <filtering-router>
          <outbound-endpoint ref="ComponentRequestQueueEndpoint"
              synchronous="false" />
          <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
              ComponentMessageType" />
        </filtering-router>
      </outbound>
    </service>
  </model>
</mule>
```

A.1.5 storage-model

Listing: storage-model.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org/schema/mule/core/2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:spring="
      http://www.springframework.org/schema/beans"
  xmlns:vm="http://www.mulesource.org/schema/mule/vm/2.0" xmlns:euca="
      http://www.eucalyptus.com/schema/cloud/1.6"
  xsi:schemaLocation="
       http://www.springframework.org/schema/beans http://www.
           springframework.org/schema/beans/spring-beans-2.0.xsd
       http://www.mulesource.org/schema/mule/core/2.0 http://www.
           mulesource.org/schema/mule/core/2.0/mule.xsd
       http://www.mulesource.org/schema/mule/vm/2.0 http://www.
           mulesource.org/schema/mule/vm/2.0/mule-vm.xsd
       http://www.eucalyptus.com/schema/cloud/1.6 http://www.eucalyptus
           .com/schema/cloud/1.6/euca.xsd">
  <model name="storage">
```

```
    <default-service-exception-strategy>
      <outbound-endpoint ref="StorageReplyQueueWS" />
    </default-service-exception-strategy>
    <service name="StorageRequestQueue">
      <inbound>
        <vm:inbound-endpoint ref="StorageRequestQueueEndpoint" />
      </inbound>
      <bridge-component />
      <outbound>
        <filtering-router>
          <outbound-endpoint ref="StorageWS" />
          <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
              StorageRequestType" />
        </filtering-router>
        <filtering-router>
          <outbound-endpoint ref="StorageWS" />
          <payload-type-filter
            expectedType="edu.ucsb.eucalyptus.msgs.
                StorageComponentMessageType" />
        </filtering-router>
      </outbound>
    </service>
    <service name="Storage">
      <inbound>
        <inbound-endpoint ref="StorageWS" />
      </inbound>
      <component class="edu.ucsb.eucalyptus.cloud.ws.BlockStorage" />
      <outbound>
        <filtering-router>
          <outbound-endpoint ref="ReplyQueueEndpoint" />
          <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
              StorageComponentMessageResponseType" />
        </filtering-router>
        <filtering-router>
          <outbound-endpoint ref="StorageReplyQueueWS" />
          <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
              StorageResponseType" />
        </filtering-router>
      </outbound>
    </service>
    <service name="StorageReplyQueue">
      <inbound>
        <inbound-endpoint ref="StorageReplyQueueWS" />
      </inbound>
      <component class="edu.ucsb.eucalyptus.ic.StorageReplyQueue" />
    </service>
  </model>
  <model name="storage-internal">
    <service name="StorageInternal">
      <inbound>
        <inbound-endpoint ref="StorageInternalWS" />
        <inbound-endpoint ref="StorageInternalVM" />
      </inbound>
```

```
      <component class="edu.ucsb.eucalyptus.cloud.ws.BlockStorage" />
    </service>
  </model>
</mule>
```

A.1.6 storage-services

Listing: storage-services.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org/schema/mule/core/2.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:vm="http://www.mulesource.org/schema/mule/vm/2.0"
      xmlns:euca="http://www.eucalyptus.com/schema/cloud/1.6"
      xsi:schemaLocation="
       http://www.springframework.org/schema/beans http://www.
           springframework.org/schema/beans/spring-beans-2.0.xsd
       http://www.mulesource.org/schema/mule/core/2.0 http://www.
           mulesource.org/schema/mule/core/2.0/mule.xsd
       http://www.mulesource.org/schema/mule/vm/2.0 http://www.
           mulesource.org/schema/mule/vm/2.0/mule-vm.xsd
       http://www.eucalyptus.com/schema/cloud/1.6 http://www.eucalyptus
           .com/schema/cloud/1.6/euca.xsd">
    <euca:endpoint name="StorageControllerWS"
                   connector-ref="eucaws"
                   address="http://127.0.0.1:${euca.ws.port}/services/
                       Storage" />
    <euca:endpoint name="StorageInternalWS"
                   connector-ref="eucaws"
                   address="http://127.0.0.1:${euca.ws.port}/internal/
                       StorageInternal"
                   synchronous="true" />
    <endpoint name="StorageWS" address="vm://Storage" synchronous="
        false" />
    <endpoint name="StorageInternalVM" address="vm://StorageInternal"
        synchronous="true" />
    <endpoint name="StorageReplyQueueWS" address="vm://
        StorageReplyQueue" synchronous="false" />
</mule>
```

A.1.7 eucalyptus-userdata

Listing: eucalyptus-userdata.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org/schema/mule/core/2.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:vm="http://www.mulesource.org/schema/mule/vm/2.0"
      xmlns:euca="http://www.eucalyptus.com/schema/cloud/1.6"
```

```
        xsi:schemaLocation="
         http://www.springframework.org/schema/beans http://www.
             springframework.org/schema/beans/spring-beans-2.0.xsd
         http://www.mulesource.org/schema/mule/core/2.0 http://www.
             mulesource.org/schema/mule/core/2.0/mule.xsd
         http://www.mulesource.org/schema/mule/vm/2.0 http://www.
             mulesource.org/schema/mule/vm/2.0/mule-vm.xsd
         http://www.eucalyptus.com/schema/cloud/1.6 http://www.eucalyptus
             .com/schema/cloud/1.6/euca.xsd">
    <model name="eucalyptus-userdata">
        <default-service-exception-strategy>
            <outbound-endpoint ref="ReplyQueueEndpoint" />
        </default-service-exception-strategy>
        <service name="KeyPair">
            <inbound>
                <inbound-endpoint ref="KeyPairWS" />
            </inbound>
            <component class="com.eucalyptus.keys.KeyPairManager" />
            <outbound>
                <outbound-pass-through-router>
                    <outbound-endpoint ref="ReplyQueueEndpoint" />
                </outbound-pass-through-router>
            </outbound>
        </service>
        <service name="Groups">
            <inbound>
                <inbound-endpoint ref="GroupsWS" />
            </inbound>
            <component class="com.eucalyptus.network.
                NetworkGroupManager" />
            <outbound>
                <outbound-pass-through-router>
                    <outbound-endpoint ref="ReplyQueueEndpoint" />
                </outbound-pass-through-router>
            </outbound>
        </service>
    </model>
</mule>
```

A.1.8 eucalyptus-storage

Listing: eucalyptus-storage.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org/schema/mule/core/2.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:vm="http://www.mulesource.org/schema/mule/vm/2.0"
      xmlns:euca="http://www.eucalyptus.com/schema/cloud/1.6"
      xsi:schemaLocation="
       http://www.springframework.org/schema/beans http://www.
           springframework.org/schema/beans/spring-beans-2.0.xsd
```

```
        http://www.mulesource.org/schema/mule/core/2.0 http://www.
            mulesource.org/schema/mule/core/2.0/mule.xsd
        http://www.mulesource.org/schema/mule/vm/2.0 http://www.
            mulesource.org/schema/mule/vm/2.0/mule-vm.xsd
        http://www.eucalyptus.com/schema/cloud/1.6 http://www.eucalyptus
            .com/schema/cloud/1.6/euca.xsd">
    <model name="eucalyptus-storage">
        <default-service-exception-strategy>
            <outbound-endpoint ref="ReplyQueueEndpoint"/>
        </default-service-exception-strategy>
        <service name="Image">
            <inbound>
                <inbound-endpoint ref="ImageWS"/>
            </inbound>
            <component class="com.eucalyptus.images.ImageManager"/>
            <outbound>
                <outbound-pass-through-router>
                    <outbound-endpoint ref="ReplyQueueEndpoint"/>
                </outbound-pass-through-router>
            </outbound>
        </service>
        <service name="Volume">
            <inbound>
                <inbound-endpoint ref="VolumeWS"/>
            </inbound>
            <component class="com.eucalyptus.blockstorage.VolumeManager
                "/>
            <outbound>
                <outbound-pass-through-router>
                    <outbound-endpoint ref="ReplyQueueEndpoint"/>
                </outbound-pass-through-router>
            </outbound>
        </service>
        <service name="Snapshot">
            <inbound>
                <inbound-endpoint ref="SnapshotWS"/>
            </inbound>
            <component class="com.eucalyptus.blockstorage.
                SnapshotManager"/>
            <outbound>
                <outbound-pass-through-router>
                    <outbound-endpoint ref="ReplyQueueEndpoint"/>
                </outbound-pass-through-router>
            </outbound>
        </service>
    </model>
</mule>
```

A.1.9 eucalyptus-services

Listing: eucalyptus-services.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org/schema/mule/core/2.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:vm="http://www.mulesource.org/schema/mule/vm/2.0"
      xmlns:euca="http://www.eucalyptus.com/schema/cloud/1.6"
      xsi:schemaLocation="
       http://www.springframework.org/schema/beans http://www.
           springframework.org/schema/beans/spring-beans-2.0.xsd
       http://www.mulesource.org/schema/mule/core/2.0 http://www.
           mulesource.org/schema/mule/core/2.0/mule.xsd
       http://www.mulesource.org/schema/mule/vm/2.0 http://www.
           mulesource.org/schema/mule/vm/2.0/mule-vm.xsd
       http://www.eucalyptus.com/schema/cloud/1.6 http://www.eucalyptus
           .com/schema/cloud/1.6/euca.xsd">
  <!--user data services-->
  <euca:endpoint name="EucalyptusWS" connector-ref="eucaws" address="
      http://127.0.0.1:${euca.ws.port}/services/Eucalyptus" synchronous=
      "true" />
  <endpoint name="ShortBusWS" address="vm://ShortBus" synchronous="
      false" />
  <endpoint name="ImageWS" address="vm://Image" synchronous="false" />
  <endpoint name="VolumeWS" address="vm://Volume" synchronous="false" /
      >
  <endpoint name="SnapshotWS" address="vm://Snapshot" synchronous="
      false" />
  <endpoint name="KeyPairWS" address="vm://KeyPair" synchronous="false"
       />
  <endpoint name="GroupsWS" address="vm://Groups" synchronous="false" /
      >
  <!--vm request verification pipeline-->
  <endpoint name="StartVerifyWS" address="vm://VmVerify" synchronous="
      false" />
  <endpoint name="ImageVerifyWS" address="vm://ImageVerify" synchronous
      ="false" />
  <endpoint name="KeyPairVerifyWS" address="vm://KeyPairVerify"
      synchronous="false" />
  <endpoint name="GroupsVerifyWS" address="vm://GroupsVerify"
      synchronous="false" />
  <endpoint name="VmTypeVerifyWS" address="vm://VmTypeVerify"
      synchronous="false" />
  <endpoint name="FinishedVerifyWS" address="vm://VmVerified"
      synchronous="false" />
  <endpoint name="KeyPairResolveWS" address="vm://KeyPairResolve"
      synchronous="true" />
  <endpoint name="ImageResolveWS" address="vm://ImageResolve"
      synchronous="true" />
  <!--run-time system state services-->
  <endpoint name="ClusterEndpointWS" address="vm://ClusterEndpoint"
      synchronous="false" />
  <endpoint name="ClusterSinkWS" address="vm://ClusterSink" synchronous
      ="false" />
```

```
  <endpoint name="VmMetadataWS" address="vm://VmMetadata" synchronous="
      true" />
  <endpoint name="VmControlWS" address="vm://VmControl" synchronous="
      false" />
  <endpoint name="AddressWS" address="vm://Address" synchronous="false"
       />
  <endpoint name="UpdateSystemWS" address="vm://UpdateSystemState"
      synchronous="false" />
  <endpoint name="SystemStateWS" address="vm://SystemState" synchronous
      ="false" />
  <endpoint name="TransformReplyWS" address="vm://TransformReply"
      synchronous="false" />
</mule>
```

A.1.10 eucalyptus-runtime

Listing: eucalyptus-runtime.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org/schema/mule/core/2.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:spring="http://www.springframework.org/schema/beans"
       xmlns:vm="http://www.mulesource.org/schema/mule/vm/2.0"
       xmlns:euca="http://www.eucalyptus.com/schema/cloud/1.6"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.
            springframework.org/schema/beans/spring-beans-2.0.xsd
        http://www.mulesource.org/schema/mule/core/2.0 http://www.
            mulesource.org/schema/mule/core/2.0/mule.xsd
        http://www.mulesource.org/schema/mule/vm/2.0 http://www.
            mulesource.org/schema/mule/vm/2.0/mule-vm.xsd
        http://www.eucalyptus.com/schema/cloud/1.6 http://www.eucalyptus
            .com/schema/cloud/1.6/euca.xsd">
  <model name="eucalyptus-runtime">
    <default-service-exception-strategy>
      <outbound-endpoint ref="ReplyQueueEndpoint"/>
    </default-service-exception-strategy>
    <service name="VmControl">
      <inbound>
        <inbound-endpoint ref="VmControlWS"/>
      </inbound>
      <component class="com.eucalyptus.vm.VmControl"/>
      <outbound>
        <outbound-pass-through-router>
          <outbound-endpoint ref="ReplyQueueEndpoint"/>
        </outbound-pass-through-router>
      </outbound>
    </service>
      <service name="KeyPairResolve">
        <inbound>
          <inbound-endpoint ref="KeyPairResolveWS"/>
        </inbound>
```

```
      <component class="com.eucalyptus.keys.KeyPairManager"/>
    </service>
    <service name="ImageResolve">
      <inbound>
        <inbound-endpoint ref="ImageResolveWS"/>
      </inbound>
      <component class="com.eucalyptus.images.ImageManager"/>
    </service>
  <service name="ClusterSink">
    <inbound>
      <inbound-endpoint ref="ClusterSinkWS"/>
    </inbound>
    <component class="com.eucalyptus.cluster.ClusterEndpoint"/>
  </service>
  <service name="ClusterEndpoint">
    <inbound>
      <inbound-endpoint ref="ClusterEndpointWS"/>
    </inbound>
    <component class="com.eucalyptus.cluster.ClusterEndpoint"/>
    <outbound>
      <outbound-pass-through-router>
        <outbound-endpoint ref="ReplyQueueEndpoint"/>
      </outbound-pass-through-router>
    </outbound>
  </service>
  <service name="Address">
    <inbound>
      <inbound-endpoint ref="AddressWS"/>
    </inbound>
    <component class="com.eucalyptus.address.AddressManager"/>
    <outbound>
      <outbound-pass-through-router>
        <outbound-endpoint ref="ReplyQueueEndpoint"/>
      </outbound-pass-through-router>
    </outbound>
  </service>
  <service name="SystemState">
    <inbound>
      <inbound-endpoint ref="SystemStateWS"/>
    </inbound>
    <component class="com.eucalyptus.vm.SystemState"/>
    <outbound>
      <outbound-pass-through-router>
        <outbound-endpoint ref="ReplyQueueEndpoint"/>
      </outbound-pass-through-router>
    </outbound>
  </service>
  <service name="UpdateSystemAndReply">
    <inbound>
      <inbound-endpoint ref="UpdateSystemWS"/>
    </inbound>
    <component class="com.eucalyptus.sla.CreateVmInstances"/>
    <outbound>
```

```
        <multicasting-router>
          <outbound-endpoint ref="ClusterSinkWS"/>
          <outbound-endpoint ref="TransformReplyWS"/>
        </multicasting-router>
      </outbound>
    </service>
    <service name="TransformReply">
      <inbound>
        <inbound-endpoint ref="TransformReplyWS"/>
      </inbound>
      <component class="com.eucalyptus.vm.VmReplyTransform"/>
      <outbound>
        <outbound-pass-through-router>
          <outbound-endpoint ref="ReplyQueueEndpoint"/>
        </outbound-pass-through-router>
      </outbound>
    </service>
  </model>
  <model name="vm-metadata">
    <service name="VmMetadata">
      <inbound>
        <inbound-endpoint ref="VmMetadataWS"/>
      </inbound>
      <component class="com.eucalyptus.vm.VmMetadata"/>
    </service>
  </model>
</mule>
```

A.1.11 eucalyptus-interface

Listing: eucalyptus-interface.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org/schema/mule/core/2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:spring="
      http://www.springframework.org/schema/beans"
  xmlns:vm="http://www.mulesource.org/schema/mule/vm/2.0" xmlns:euca="
      http://www.eucalyptus.com/schema/cloud/1.6"
  xsi:schemaLocation="
       http://www.springframework.org/schema/beans http://www.
           springframework.org/schema/beans/spring-beans-2.0.xsd
       http://www.mulesource.org/schema/mule/core/2.0 http://www.
           mulesource.org/schema/mule/core/2.0/mule.xsd
       http://www.mulesource.org/schema/mule/vm/2.0 http://www.
           mulesource.org/schema/mule/vm/2.0/mule-vm.xsd
       http://www.eucalyptus.com/schema/cloud/1.6 http://www.eucalyptus
           .com/schema/cloud/1.6/euca.xsd">
  <model name="eucalyptus-interface">
    <default-service-exception-strategy>
      <outbound-endpoint ref="ReplyQueueEndpoint" />
    </default-service-exception-strategy>
    <service name="EucalyptusRequestQueue">
```

```
<inbound>
  <inbound-endpoint ref="EucalyptusRequestQueueEndpoint"/>
</inbound>
<bridge-component/>
<outbound>
  <filtering-router>
    <outbound-endpoint ref="ShortBusWS" />
    <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
        DescribeRegionsType" />
  </filtering-router>
  <filtering-router>
    <outbound-endpoint ref="StartVerifyWS" />
    <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
        RunInstancesType" />
  </filtering-router>
  <filtering-router>
    <outbound-endpoint ref="VmControlWS" />
    <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
        VmControlMessage" />
  </filtering-router>
  <filtering-router>
    <outbound-endpoint ref="VmControlWS" />
    <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
        VmBundleMessage" />
  </filtering-router>
  <filtering-router>
    <outbound-endpoint ref="ClusterEndpointWS" />
    <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
        ClusterMessage" />
  </filtering-router>
  <filtering-router>
    <outbound-endpoint ref="KeyPairWS" />
    <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
        VmKeyPairMessage" />
  </filtering-router>
  <filtering-router>
    <outbound-endpoint ref="ImageWS" />
    <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
        VmImageMessage" />
  </filtering-router>
  <filtering-router>
    <outbound-endpoint ref="VolumeWS" />
    <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
        BlockVolumeMessage" />
  </filtering-router>
  <filtering-router>
    <outbound-endpoint ref="SnapshotWS" />
    <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
        BlockSnapshotMessage" />
  </filtering-router>
  <filtering-router>
    <outbound-endpoint ref="AddressWS" />
```

```
        <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
            VmAddressMessage" />
      </filtering-router>
      <filtering-router>
        <outbound-endpoint ref="GroupsWS" />
        <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
            VmSecurityMessage" />
      </filtering-router>
      <filtering-router>
        <outbound-endpoint ref="ShortBusWS" />
        <payload-type-filter expectedType="edu.ucsb.eucalyptus.msgs.
            EucalyptusMessage" />
      </filtering-router>
    </outbound>
  </service>
    <service name="ShortBus">
        <inbound>
        <inbound-endpoint ref="ShortBusWS" />
        </inbound>
        <component class="edu.ucsb.eucalyptus.ic.Eucalyptus" />
        <outbound>
        <outbound-pass-through-router>
          <outbound-endpoint ref="ReplyQueueEndpoint" />
          </outbound-pass-through-router>
        </outbound>
    </service>
  <service name="Eucalyptus">
    <inbound>
      <inbound-endpoint ref="EucalyptusWS" />
    </inbound>
    <component class="edu.ucsb.eucalyptus.ic.Eucalyptus" />
  </service>
 </model>
</mule>
```

A.2 XML Configuration Files of Axis2/C (Eucalyptus Cloud)

The XML files below provide the configuration of provenance to the Cluster and Node Controller services of the Eucalyptus Cloud.

A.2.1 EucalyptusCC

The *XML* file below provides the configuration of provenance to the various methods of the Eucalyptus Cluster Controller service. The provenance collection can be enabled/disabled for a particular Cluster and its various methods.

Listing: services.xml (Eucalytus CC)

```
<?xml version="1.0" encoding="UTF-8"?>
<service name="EucalyptusCC">
```

```
<parameter name="wsdl_path" locked="xsd:false">/opt/eucalyptus/packages
    /axis2c-1.6.0/services/EucalyptusCC/eucalyptus_cc.wsdl</parameter>
<parameter name="ServiceClass">EucalyptusCC</parameter>
<description>EucalyptusCC Service
            </description>
<operation name="DescribeInstances">
<parameter name="wsamapping">EucalyptusCC#DescribeInstances</parameter>
</operation>
<operation name="ConfigureNetwork">
<parameter name="wsamapping">EucalyptusCC#ConfigureNetwork</parameter>
</operation>
<operation name="DescribeResources">
<parameter name="wsamapping">EucalyptusCC#DescribeResources</parameter>
</operation>
<operation name="StartNetwork">
<parameter name="wsamapping">EucalyptusCC#StartNetwork</parameter>
</operation>
<operation name="StopNetwork">
<parameter name="wsamapping">EucalyptusCC#StopNetwork</parameter>
</operation>
<operation name="DescribeNetworks">
<parameter name="wsamapping">EucalyptusCC#DescribeNetworks</parameter>
</operation>
<operation name="AssignAddress">
<parameter name="wsamapping">EucalyptusCC#AssignAddress</parameter>
</operation>
<operation name="DescribePublicAddresses">
<parameter name="wsamapping">EucalyptusCC#DescribePublicAddresses</
    parameter>
</operation>
<operation name="RebootInstances">
<parameter name="wsamapping">EucalyptusCC#RebootInstances</parameter>
</operation>
<operation name="GetConsoleOutput">
<parameter name="wsamapping">EucalyptusCC#GetConsoleOutput</parameter>
</operation>
<operation name="UnassignAddress">
<parameter name="wsamapping">EucalyptusCC#UnassignAddress</parameter>
</operation>
<operation name="TerminateInstances">
<parameter name="wsamapping">EucalyptusCC#TerminateInstances</parameter
    >
</operation>
<operation name="DetachVolume">
<parameter name="wsamapping">EucalyptusCC#DetachVolume</parameter>
</operation>
<operation name="AttachVolume">
<parameter name="wsamapping">EucalyptusCC#AttachVolume</parameter>
</operation>
<operation name="RunInstances">
<parameter name="wsamapping">EucalyptusCC#RunInstances</parameter>
</operation>
<module ref="rampart"/>
```

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:AsymmetricBinding xmlns:sp="http://schemas.xmlsoap.org/ws
          /2005/07/securitypolicy">
  <wsp:Policy>
          <sp:InitiatorToken>
            <wsp:Policy>
              <sp:X509Token sp:IncludeToken="http://schemas.xmlsoap.org
                  /ws/2005/07/securitypolicy/IncludeToken/Always">a
                <wsp:Policy>
                  <sp:RequireEmbeddedTokenReference/>
                  <sp:WssX509V3Token10/>
                </wsp:Policy>
              </sp:X509Token>
            </wsp:Policy>
          </sp:InitiatorToken>
          <sp:RecipientToken>
            <wsp:Policy>
              <sp:X509Token sp:IncludeToken="http://schemas.xmlsoap.org
                  /ws/2005/07/securitypolicy/IncludeToken/Always">
                <wsp:Policy>
                  <sp:RequireEmbeddedTokenReference/>
                  <sp:WssX509V3Token10/>
                </wsp:Policy>
              </sp:X509Token>
            </wsp:Policy>
          </sp:RecipientToken>

          <sp:AlgorithmSuite>
            <wsp:Policy>
              <sp:Basic256Rsa15/>
            </wsp:Policy>
          </sp:AlgorithmSuite>

          <sp:Layout>
            <wsp:Policy>
              <sp:Strict/>
            </wsp:Policy>
          </sp:Layout>

    <sp:IncludeTimestamp/>
          <sp:OnlySignEntireHeadersAndBody/>
    <!-- <sp:EncryptSignature/> -->
        </wsp:Policy>
      </sp:AsymmetricBinding>

      <sp:Wss10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/
          securitypolicy">
        <wsp:Policy>
          <sp:MustSupportRefKeyIdentifier/>
          <sp:MustSupportRefEmbeddedToken/>
    <sp:MustSupportRefIssuerSerial/>
```

```
        </wsp:Policy>
      </sp:Wss10>

      <sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/
          securitypolicy">
  <sp:Body/>
        <sp:Header Namespace="http://www.w3.org/2005/08/addressing"/>
      </sp:SignedParts>

      <rampc:RampartConfig xmlns:rampc="http://ws.apache.org/rampart/c/
          policy">
  <rampc:ReceiverCertificate>/opt/eucalyptus/var/lib/eucalyptus/keys/
      cloud-cert.pem</rampc:ReceiverCertificate>
  <rampc:Certificate>/opt/eucalyptus/var/lib/eucalyptus/keys/cluster-
      cert.pem</rampc:Certificate>
  <rampc:PrivateKey>/opt/eucalyptus/var/lib/eucalyptus/keys/cluster-pk.
      pem</rampc:PrivateKey>
  <!-- <rampc:TimeToLive>14400</rampc:TimeToLive> -->
  <rampc:ClockSkewBuffer>20</rampc:ClockSkewBuffer>
      </rampc:RampartConfig>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>

</service>
```

A.2.2 EucalyptusNC

The *XML* file below provides the configuration of provenance to the various methods of the Eucalyptus Node Controller service. The provenance collection can be enabled/disabled for a particular Node and its various methods.

Listing: servicex.xml (EucalyptusNC)

```
<?xml version="1.0" encoding="UTF-8"?>
<service name="EucalyptusNC">
<parameter name="wsdl_path" locked="xsd:false">/opt/eucalyptus/packages
    /axis2c-1.6.0/services/EucalyptusNC/eucalyptus_nc.wsdl</parameter>
<parameter name="ServiceClass">EucalyptusNC</parameter>
<description>EucalyptusNC Service
            </description>
<operation name="ncRunInstance">
<parameter name="wsamapping">EucalyptusNC#ncRunInstance</parameter>
</operation>
<operation name="ncRebootInstance">
<parameter name="wsamapping">EucalyptusNC#ncRebootInstance</parameter>
</operation>
<operation name="ncGetConsoleOutput">
<parameter name="wsamapping">EucalyptusNC#ncGetConsoleOutput</parameter
    >
</operation>
<operation name="ncDetachVolume">
<parameter name="wsamapping">EucalyptusNC#ncDetachVolume</parameter>
```

```
</operation>
<operation name="ncDescribeInstances">
<parameter name="wsamapping">EucalyptusNC#ncDescribeInstances</
    parameter>
</operation>
<operation name="ncAttachVolume">
<parameter name="wsamapping">EucalyptusNC#ncAttachVolume</parameter>
</operation>
<operation name="ncPowerDown">
<parameter name="wsamapping">EucalyptusNC#ncPowerDown</parameter>
</operation>
<operation name="ncDescribeResource">
<parameter name="wsamapping">EucalyptusNC#ncDescribeResource</parameter
    >
</operation>
<operation name="ncTerminateInstance">
<parameter name="wsamapping">EucalyptusNC#ncTerminateInstance</
    parameter>
</operation>
<operation name="ncStartNetwork">
<parameter name="wsamapping">EucalyptusNC#ncStartNetwork</parameter>
</operation>
<module ref="rampart"/>
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:AsymmetricBinding xmlns:sp="http://schemas.xmlsoap.org/ws
          /2005/07/securitypolicy">
  <wsp:Policy>
          <sp:InitiatorToken>
            <wsp:Policy>
              <sp:X509Token sp:IncludeToken="http://schemas.xmlsoap.org
                  /ws/2005/07/securitypolicy/IncludeToken/Always">a
                <wsp:Policy>
                  <sp:RequireEmbeddedTokenReference/>
                  <sp:WssX509V3Token10/>
                </wsp:Policy>
              </sp:X509Token>
            </wsp:Policy>
          </sp:InitiatorToken>
          <sp:RecipientToken>
            <wsp:Policy>
              <sp:X509Token sp:IncludeToken="http://schemas.xmlsoap.org
                  /ws/2005/07/securitypolicy/IncludeToken/Always">
                <wsp:Policy>
                  <sp:RequireEmbeddedTokenReference/>
                  <sp:WssX509V3Token10/>
                </wsp:Policy>
              </sp:X509Token>
            </wsp:Policy>
          </sp:RecipientToken>

          <sp:AlgorithmSuite>
```

```
            <wsp:Policy>
              <sp:Basic256Rsa15/>
            </wsp:Policy>
          </sp:AlgorithmSuite>

          <sp:Layout>
            <wsp:Policy>
              <sp:Strict/>
            </wsp:Policy>
          </sp:Layout>

    <sp:IncludeTimestamp/>
          <sp:OnlySignEntireHeadersAndBody/>
    <!-- <sp:EncryptSignature/> -->
        </wsp:Policy>
      </sp:AsymmetricBinding>

      <sp:Wss10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/
          securitypolicy">
        <wsp:Policy>
          <sp:MustSupportRefKeyIdentifier/>
          <sp:MustSupportRefEmbeddedToken/>
    <sp:MustSupportRefIssuerSerial/>
        </wsp:Policy>
      </sp:Wss10>

      <sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/
          securitypolicy">
  <sp:Body/>
        <sp:Header Namespace="http://www.w3.org/2005/08/addressing"/>
      </sp:SignedParts>

      <rampc:RampartConfig xmlns:rampc="http://ws.apache.org/rampart/c/
          policy">
  <rampc:ReceiverCertificate>/opt/eucalyptus/var/lib/eucalyptus/keys/
      cluster-cert.pem</rampc:ReceiverCertificate>
  <rampc:Certificate>/opt/eucalyptus/var/lib/eucalyptus/keys/node-cert.
      pem</rampc:Certificate>
  <rampc:PrivateKey>/opt/eucalyptus/var/lib/eucalyptus/keys/node-pk.pem
      </rampc:PrivateKey>
  <!-- <rampc:TimeToLive>14400</rampc:TimeToLive> -->
  <rampc:ClockSkewBuffer>20</rampc:ClockSkewBuffer>
      </rampc:RampartConfig>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
</service>
```

A.2.3 Eucalyptus-Axis2/C

The *XML* file below provides the configuration of provenance into Axis2/C with various phases. The provenance collection can be enabled/disabled for all the Nodes and Clusters globally by editing

the file below.

Listing: axis2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<axisconfig name="AxisJava2.0">
    <parameter name="sendStacktraceDetailsWithFaults">false</parameter>
    <parameter name="DrillDownToRootCauseForFaultReason">false</
        parameter>
    <parameter name="userName">admin</parameter>
    <parameter name="password">axis2</parameter>
    <parameter name="disableREST" locked="true">false</parameter>
    <parameter name="disableSOAP12" locked="true">false</parameter>
    <deployer extension=".class" directory="pojo" class="org.apache.
        axis2.deployment.POJODeployer"/>
    <deployer extension=".jar" directory="servicejars" class="org.
        apache.axis2.jaxws.framework.JAXWSDeployer"/>
    <threadContextMigrators>
        <threadContextMigrator listId="JAXWS-ThreadContextMigrator-List
            " class="org.apache.axis2.jaxws.addressing.migrator.
            EndpointContextMapMigrator"/>
    </threadContextMigrators>
     <messageReceivers>
        <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
                         class="org.apache.axis2.receivers.
                             RawXMLINOnlyMessageReceiver"/>
        <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
                         class="org.apache.axis2.receivers.
                             RawXMLINOutMessageReceiver"/>
        <messageReceiver mep="http://www.w3.org/2006/01/wsdl/in-only"
                         class="org.apache.axis2.receivers.
                             RawXMLINOnlyMessageReceiver"/>
        <messageReceiver mep="http://www.w3.org/2006/01/wsdl/in-out"
                         class="org.apache.axis2.receivers.
                             RawXMLINOutMessageReceiver"/>
    </messageReceivers>
    <messageFormatters>
        <messageFormatter contentType="application/x-www-form-
            urlencoded"
                         class="org.apache.axis2.transport.http.
                             XFormURLEncodedFormatter"/>
        <messageFormatter contentType="multipart/form-data"
                         class="org.apache.axis2.transport.http.
                             MultipartFormDataFormatter"/>
        <messageFormatter contentType="application/xml"
                         class="org.apache.axis2.transport.http.
                             ApplicationXMLFormatter"/>
        <messageFormatter contentType="text/xml"
                         class="org.apache.axis2.transport.http.
                             SOAPMessageFormatter"/>
        <messageFormatter contentType="application/soap+xml"
                         class="org.apache.axis2.transport.http.
                             SOAPMessageFormatter"/>
    </messageFormatters>
```

```
<messageBuilders>
    <messageBuilder contentType="application/xml"
                    class="org.apache.axis2.builder.
                        ApplicationXMLBuilder"/>
    <messageBuilder contentType="application/xml"
                    class="org.apache.axis2.builder.
                        ApplicationXMLBuilder"/>
    <messageBuilder contentType="application/x-www-form-urlencoded"
                    class="org.apache.axis2.builder.
                        XFormURLEncodedBuilder"/>
    <messageBuilder contentType="multipart/form-data"
                    class="org.apache.axis2.builder.
                        MultipartFormDataBuilder"/>
</messageBuilders>

<transportReceiver name="http"
                   class="org.apache.axis2.transport.http.
                       SimpleHTTPServer">
    <parameter name="port">8080</parameter>
</transportReceiver>

<transportReceiver name="jms" class="org.apache.axis2.transport.jms
    .JMSListener">
    <parameter name="myTopicConnectionFactory">
      <parameter name="java.naming.factory.initial">org.apache.
          activemq.jndi.ActiveMQInitialContextFactory</parameter>
      <parameter name="java.naming.provider.url">tcp://
          localhost:61616</parameter>
      <parameter name="transport.jms.ConnectionFactoryJNDIName">
          TopicConnectionFactory</parameter>
    </parameter>

    <parameter name="myQueueConnectionFactory">
      <parameter name="java.naming.factory.initial">org.apache.
          activemq.jndi.ActiveMQInitialContextFactory</parameter>
      <parameter name="java.naming.provider.url">tcp://
          localhost:61616</parameter>
      <parameter name="transport.jms.ConnectionFactoryJNDIName">
          QueueConnectionFactory</parameter>
    </parameter>
    <parameter name="default">
      <parameter name="java.naming.factory.initial">org.apache.
          activemq.jndi.ActiveMQInitialContextFactory</parameter>
      <parameter name="java.naming.provider.url">tcp://
          localhost:61616</parameter>
      <parameter name="transport.jms.ConnectionFactoryJNDIName">
          QueueConnectionFactory</parameter>
    </parameter>
</transportReceiver>-->

<transportReceiver name="http" class="org.apache.axis2.transport.
    nhttp.HttpCoreNIOListener">
```

```
    <parameter name="port" locked="false">9000</parameter>
    <parameter name="non-blocking" locked="false">true</parameter>
  </transportReceiver>-->

  <transportReceiver name="https" class="org.apache.axis2.transport.
      nhttp.HttpCoreNIOSSLListener">
    <parameter name="port" locked="false">9002</parameter>
    <parameter name="non-blocking" locked="false">true</parameter>
      <parameter name="keystore" locked="false">
          <KeyStore>
              <Location>identity.jks</Location>
              <Type>JKS</Type>
              <Password>password</Password>
              <KeyPassword>password</KeyPassword>
          </KeyStore>
      </parameter>
      <parameter name="truststore" locked="false">
          <TrustStore>
              <Location>trust.jks</Location>
              <Type>JKS</Type>
              <Password>password</Password>
          </TrustStore>
      </parameter>-->
  <transportSender name="tcp"
                   class="org.apache.axis2.transport.tcp.
                       TCPTransportSender" />
  <transportSender name="local"
                   class="org.apache.axis2.transport.local.
                       LocalTransportSender" />
  <transportSender name="http"
                   class="org.apache.axis2.transport.http.
                       CommonsHTTPTransportSender">
      <parameter name="PROTOCOL">HTTP/1.1</parameter>
      <parameter name="Transfer-Encoding">chunked</parameter>
  </transportSender>
  <transportSender name="https"
                   class="org.apache.axis2.transport.http.
                       CommonsHTTPTransportSender">
      <parameter name="PROTOCOL">HTTP/1.1</parameter>
      <parameter name="Transfer-Encoding">chunked</parameter>
  </transportSender>

  <!-- ================================================= -->
  <!-- Global Modules -->
  <!-- ================================================= -->
  <!-- Comment this to disable Addressing -->
  <module ref="addressing" />
  <!--Configuring module , providing parameters for modules whether
      they refer or not-->
  <!--<moduleConfig name="addressing">-->
  <!--<parameter name="addressingPara">N/A</parameter>-->
  <!--</moduleConfig>-->
```

```
<!-- Clustering -->
<cluster class="org.apache.axis2.cluster.tribes.
    TribesClusterManager">
    <parameter name="param1">value1</parameter>
    <parameter name="domain">apache.axis2.domain</parameter>
    <parameter name="synchronizeAll">true</parameter>
    <parameter name="maxRetries">10</parameter>
  <configurationManager class="org.apache.axis2.cluster.
      configuration.TribesConfigurationManager">
      <listener class="org.apache.axis2.cluster.configuration.
          DefaultConfigurationManagerListener"/>
  </configurationManager>
  <contextManager class="org.apache.axis2.cluster.context.
      TribesContextManager">
      <listener class="org.apache.axis2.cluster.context.
          DefaultContextManagerListener"/>
  </contextManager>
</cluster>
-->

<!-- Phases -->

<phaseOrder type="InFlow">
    <!-- System predefined phases -->
    <phase name="Transport">
        <handler name="RequestURIBasedDispatcher"
                 class="org.apache.axis2.dispatchers.
                     RequestURIBasedDispatcher">
            <order phase="Transport"/>
        </handler>
        <handler name="SOAPActionBasedDispatcher"
                 class="org.apache.axis2.dispatchers.
                     SOAPActionBasedDispatcher">
            <order phase="Transport"/>
        </handler>
    </phase>
    <phase name="Addressing">
         <handler name="AddressingBasedDispatcher"
                 class="org.apache.axis2.dispatchers.
                     AddressingBasedDispatcher">
             <order phase="Addressing"/>
        </handler>
    </phase>
    <phase name="Security"/>
    <phase name="PreDispatch"/>
    <phase name="Dispatch" class="org.apache.axis2.engine.
        DispatchPhase">
        <handler name="RequestURIBasedDispatcher"
                 class="org.apache.axis2.dispatchers.
                     RequestURIBasedDispatcher"/>
        <handler name="SOAPActionBasedDispatcher"
                 class="org.apache.axis2.dispatchers.
                     SOAPActionBasedDispatcher"/>
```

```
            <handler name="RequestURIOperationDispatcher"
                     class="org.apache.axis2.dispatchers.
                         RequestURIOperationDispatcher"/>
            <handler name="SOAPMessageBodyBasedDispatcher"
                     class="org.apache.axis2.dispatchers.
                         SOAPMessageBodyBasedDispatcher"/>
            <handler name="HTTPLocationBasedDispatcher"
                     class="org.apache.axis2.dispatchers.
                         HTTPLocationBasedDispatcher"/>
            <handler name="GenericProviderDispatcher"
                     class="org.apache.axis2.jaxws.dispatchers.
                         GenericProviderDispatcher"/>
            <handler name="MustUnderstandValidationDispatcher"
                     class="org.apache.axis2.jaxws.dispatchers.
                         MustUnderstandValidationDispatcher"/>
        </phase>
        <phase name="RMPhase"/>
        <!--  System predefined phases       -->
        <!--   After Postdispatch phase module author or service author
             can add any phase he want      -->
        <phase name="OperationInPhase">
            <handler name="MustUnderstandChecker"
                     class="org.apache.axis2.jaxws.dispatchers.
                         MustUnderstandChecker">
                <order phase="OperationInPhase"/>
            </handler>
        </phase>
        <phase name="soapmonitorPhase"/>
    </phaseOrder>
    <phaseOrder type="OutFlow">
        <!--       user can add his own phases to this area  -->
        <phase name="soapmonitorPhase"/>
        <phase name="OperationOutPhase"/>
        <!--system predefined phase-->
        <!--these phase will run irrespective of the service-->
        <phase name="RMPhase"/>
        <phase name="PolicyDetermination"/>
        <phase name="MessageOut"/>
        <phase name="Security"/>
    </phaseOrder>
    <phaseOrder type="InFaultFlow">
        <phase name="Addressing">
            <handler name="AddressingBasedDispatcher"
                     class="org.apache.axis2.dispatchers.
                         AddressingBasedDispatcher">
                <order phase="Addressing"/>
            </handler>
        </phase>
        <phase name="Security"/>
        <phase name="PreDispatch"/>
        <phase name="Dispatch" class="org.apache.axis2.engine.
            DispatchPhase">
            <handler name="RequestURIBasedDispatcher"
```

```
                    class="org.apache.axis2.dispatchers.
                        RequestURIBasedDispatcher"/>
            <handler name="SOAPActionBasedDispatcher"
                    class="org.apache.axis2.dispatchers.
                        SOAPActionBasedDispatcher"/>
            <handler name="RequestURIOperationDispatcher"
                    class="org.apache.axis2.dispatchers.
                        RequestURIOperationDispatcher"/>
            <handler name="SOAPMessageBodyBasedDispatcher"
                    class="org.apache.axis2.dispatchers.
                        SOAPMessageBodyBasedDispatcher"/>
            <handler name="HTTPLocationBasedDispatcher"
                    class="org.apache.axis2.dispatchers.
                        HTTPLocationBasedDispatcher"/>
            <handler name="GenericProviderDispatcher"
                    class="org.apache.axis2.jaxws.dispatchers.
                        GenericProviderDispatcher"/>
            <handler name="MustUnderstandValidationDispatcher"
                    class="org.apache.axis2.jaxws.dispatchers.
                        MustUnderstandValidationDispatcher"/>
        </phase>
        <phase name="RMPhase"/>
        <!--       user can add his own phases to this area  -->
        <phase name="OperationInFaultPhase"/>
        <phase name="soapmonitorPhase"/>
    </phaseOrder>
    <phaseOrder type="OutFaultFlow">
        <!--       user can add his own phases to this area  -->
        <phase name="soapmonitorPhase"/>
        <phase name="OperationOutFaultPhase"/>
        <phase name="RMPhase"/>
        <phase name="PolicyDetermination"/>
        <phase name="MessageOut"/>
        <phase name="Security"/>
    </phaseOrder>
</axisconfig>
```

Curriculum Vitae

Personal Information

Name: Muhammad Imran
Birth Date: 25^{th} march 1982
Nationality: Pakistan
Address: Diefenbachgasse 46/27, 1150 Wien, Austria

Education

PhD in Computer Science, University of Vienna, Austia.
anticipated graduation: 2014

MSc in Computer Science, Quaid-i-Azam University, Islamabad, Pakistan - 2005

BSc in Mathematics and Statistics, University of Peshawar, Pakistan - 2003

Research

Currently pursuing research in the area of Cloud based platforms, Service Oriented Architectures, and Provenance. My PhD title is: **Provenance in Clouds: Framework, Applications and Implication**. This work focuses on the layered and abstract architecture of Cloud platforms, an effective framework addressing various requirements for the collection and management of provenance, and various applications using provenance ensuring the improved management of Clouds for end users. During my PhD studies, following research papers were published.

- "Searching in Cloud Object Storage by Using a Metadata Model", Imran, Muhammad and Hlavacs, Helmut In: The 9th International Conference on Semantics, Knowledge and Grids (SKG2013).
- "Layering of the Provenance Data for Cloud Computing", Imran, Muhammad and Hlavacs, Helmut, In: 8th International Conference, GPC 2013 and Colocated Workshops, Seoul, Korea, May 9-11, 2013, Grid and Pervasive Computing, pp 48-58, Lecture Notes in Computer Science, Springer Berlin Heidelberg.

- "Provenance Framework for the Cloud Infrastructure: Why and How?", Imran, Muhammad and Hlavacs, Helmut In: International Journal On Advances in Intelligent Systems, volume 6, numbers 1 and 2, 2013.
- "Applications of Provenance Data for Cloud Infrastructure", Imran, Muhammad and Hlavacs, Helmut In: The 8th International Conference on Semantics, Knowledge and Grids (SKG2012), Pages 16-23, IEEE Computer Society.
- "Provenance Framework for the Cloud Environment (IaaS)", Imran, Muhammad and Hlavacs, Helmut In: CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization: Nice, France, 2012.
- "Provenance in the Cloud: Why and How?", Imran, Muhammad and Hlavacs, Helmut In: CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization: Nice, France. **We Won the Best Paper Award.**
- "On using provenance data to increase the reliability of ubiquitous computing environments", Imran, Muhammad and Hummel, Karin Anna In: Proceedings of the 10th International Conference on Information Integration and Web-based Applications and Services Pages 547-550 : Linz, Austria. 2008, ACM.

Achievements

- **Best Paper Award,** The Third International Conference on Cloud Computing, GRIDs, and Virtualization. Nice, France, 2012.
- **HEC scholarship for PhD,** in University of Vienna, Austria.

Work Experience

Quality Assurance Engineer in Ultimus Pakistan. (2006 to 2007)

Software Engineer at Goldmine Software (2005 to 2006)

Publications

During my PhD studies, following research papers were published.

- "On using provenance data to increase the reliability of ubiquitous computing environments", Imran, Muhammad and Hummel, Karin Anna In: Proceedings of the 10th International Conference on Information Integration and Web-based Applications and Services Pages 547-550 : Linz, Austria. 2008, ACM.

  This paper presented the concept of using provenance as a reliability tool in ubiquitous environments. The focus of this paper was to establish: (i) which provenance items are significant, (ii) where should provenance be stored, and (iii) how the additional data can be kept low in the system.

- "Provenance in the Cloud: Why and How?", Imran, Muhammad and Hlavacs, Helmut In: CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization: Nice, France. **We Won the Best Paper Award.**

  This paper investigated provenance in Clouds through two key questions, i.e., *why* and *how*. The *why* part presented the reasoning for incorporating provenance in Clouds, i.e., implication of provenance enabled Clouds. Various requirements of provenance collection and management were identified for the distributed and abstract architecture of Clouds. Key provenance items were also identified for Cloud architectures. The *how* part presented the approach for the collection of provenance while addressing various requirements. These concepts are included in Chapter 3 and Chapter 4.

- "Provenance Framework for the Cloud Environment (IaaS)", Imran, Muhammad and Hlavacs, Helmut In: CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization: Nice, France, 2012.

  This paper explored the underlying architecture of Clouds, e.g., IaaS model and presented the design of the framework for the collection and management of provenance in distributed Clouds. The framework is divided into various independent services/modules. This paper also investigated the existing schemes of provenance from various fields of scientific and data computation. The existing schemes were categorized accordingly and their impact was presented assuming they are incorporated in Clouds. A use case of provenance usage

and example metadata from IaaS Cloud was also presented. These concepts are included in Chapter 3 and Chapter 4.

- "Provenance Framework for the Cloud Infrastructure: Why and How?", Imran, Muhammad and Hlavacs, Helmut In: International Journal On Advances in Intelligent Systems, volume 6, numbers 1 and 2, 2013.

  This paper extended the previous two papers and provided various implementation and evaluation results. Findings of this paper are included in Chapter 3, Chapter 4 and Chapter 7.

- "Applications of Provenance Data for Cloud Infrastructure", Imran, Muhammad and Hlavacs, Helmut In: The 8th International Conference on Semantics, Knowledge and Grids (SKG2012), Pages 16-23, IEEE Computer Society.

  This paper presented the utility of provenance in Clouds utilizing the collected provenance information and services of the framework. This was established through validating various applications scenarios which highlight the significance of provenance and the developed framework at various layers of Clouds for the different end users. The applications themselves cover a broad range of domains such as finding similarity pattern, failure tracking and efficient utilization of resources. These concepts and related findings are included in Chapter 5 and Chapter 6.

- "Searching in Cloud Object Storage by Using a Metadata Model", Imran, Muhammad and Hlavacs, Helmut In: The 9th International Conference on Semantics, Knowledge and Grids (SKG2013).

  This paper was focused on the utilization of the developed framework, its various components, and the collected provenance for object storage in Clouds. A novel application which provides the ability of provenance based search in Clouds object storage was developed. Moreover, various evaluations were performed depicting a solid performance of the application and components of the framework. These concepts and related findings are included in Chapter 5.

- "Layering of the Provenance Data for Cloud Computing", Imran, Muhammad and Hlavacs, Helmut, In: 8th International Conference, GPC 2013 and Colocated Workshops, Seoul, Korea, May 9-11, 2013, Grid and Pervasive Computing, pp 48-58, Lecture Notes in Computer Science, Springer Berlin Heidelberg.

  This paper investigated the aggregation of provenance for the layered and modular architecture of Clouds. Provenance information from the individual tiers/layers, i.e., software, platform and infrastructure were explored and presented. Hereby, the significance of aggregating the provenance from these layers was established and validated using various scenarios. These concepts are included in Chapter 6.

Bibliography

[1] Amazon elastic compute cloud. http://aws.amazon.com/ec2/. retrieved: Aug 2nd, 2013.

[2] Amazon s3. http://aws.amazon.com/s3/. retrieved: Aug 2nd, 2013.

[3] Amazon simple workflow service. http://aws.amazon.com/swf/. retrieved: Aug 2nd, 2013.

[4] Apache axis2/c manual. http://axis.apache.org/axis2/c/rampart/docs/rampartc_manual.html. [retrieved: may, 2012].

[5] Axis2- ws-addressing implementation. http://axis.apache.org/axis2/java/core/modules/addressing/index.html. [retrieved: may, 2012].

[6] Eucalyptus iaas. http://www.eucalyptus.com/eucalyptus-cloud/iaas. retrieved: Aug 2nd, 2013.

[7] Eucalyptus walrus. http://www.eucalyptus.com/eucalyptus-cloud/iaas/architecturewalrus. retrieved: Aug 2nd, 2013.

[8] M/gateway. http://gradvs1.mgateway.com/main/. retrieved: Aug 2nd, 2013.

[9] Microsoft skydrive. http://skydrive.live.com/. retrieved: Aug 2nd, 2013.

[10] Mule esb. http://www.mulesoft.org/what-mule-esb. retrieved: Aug 2nd, 2013.

[11] salesforce. http://www.salesforce.com/. retrieved: Aug 2nd, 2013.

[12] Wso2 platforms. http://wso2.com/platforms. retrieved: Aug 2nd, 2013.

[13] Xen project. http://www.xenproject.org/. retrieved: Aug 2nd, 2013.

[14] Oasis reference model for service oriented architecture 1.0. https://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf, October 2006.

[15] Cloud Computing Use Cases Whitepaper. http://www.scribd.com/doc/17929394/Cloud-Computing-Use-Cases-Whitepaper, August 2009.

[16] The future of cloud computing: opportunities fro european cloud computing beyond 2010. In Keith Jeffery and Burkhard Neidecker-Lutz, editors, *The future of cloud computing*, Brussel, 2010. European Commission: Information Society and Media.

[17] J. H. Abawajy. Fault-tolerant scheduling policy for grid computing systems. *Parallel and Distributed Processing Symposium, International*, 14:238b, 2004.

[18] Imad M. Abbadi and John Lyle. Challenges for provenance in cloud computing. In *TaPP 2011: Proceedings of the Third USENIX Workshop on the Theory and Practice of Provenance*. USENIX, 2011.

[19] Divyakant Agrawal, Sudipto Das, and Amr El Abbadi. Big data and cloud computing: current state and future opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 530–533, New York, NY, USA, 2011. ACM.

[20] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: an extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pages 423–424, June 2004.

[21] Ilkay Altintas, Oscar Barney, and Efrat Jaeger-frank. Provenance collection support in the kepler scientific workflow system. In *In Proceedings of the International Provenance and Annotation Workshop (IPAW)*, pages 118–132. Springer-Verlag, 2006.

[22] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, University of California at Berkeley, February 2009.

[23] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[24] Jayant Baliga, Robert W. A. Ayre, Kerry Hinton, and Rodney S. Tucker. Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport. *Proceedings of the IEEE*, 99(1):149–167, January 2011.

[25] Roger S. Barga, Yogesh L. Simmhan, Eran Chinthaka, Satya Sanket Sahoo, Jared Jackson, and Nelson Araujo. Provenance for scientific workflows towards reproducible research. *IEEE Data Eng. Bull.*, 33(3):50–58, 2010.

[26] Douglas K. Barry and Patrick J. Gannon. *Web Services and Service-Oriented Architecture: The Savvy Manager's Guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[27] Philip A. Bernstein. Middleware: A model for distributed system services. *Commun. ACM*, 39(2):86–98, February 1996.

[28] Deepavali Bhagwat, Laura Chiticariu, Wang chiew Tan, and Gaurav Vijayvargiya. An annotation management system for relational databases. In *In VLDB*, pages 900–911. Morgan Kaufmann, 2004.

[29] Tekin Bicer, David Chiu, and Gagan Agrawal. A framework for data-intensive computing with cloud bursting. In *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, CLUSTER '11, pages 169–177, Washington, DC, USA, 2011. IEEE Computer Society.

[30] Rajendra Bose and James Frew. Composing lineage metadata with xml for custom satellite-derived data products. In *in SSDBM*, pages 275–284, 2004.

[31] Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv.*, 37(1):1–28, March 2005.

[32] Francisco V Brasileiro, Paul D. Ezhilchelvan, Santosh K Shrivastava, Neil A Speirs, and Sha Tao. Implementing fail-silent nodes for distributed systems. *Computers, IEEE Transactions on*, 45(11):1226–1238, 1996.

[33] John Bresnahan, Kate Keahey, David LaBissoniere, and Tim Freeman. Cumulus: an open source storage cloud for science. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, pages 25–32. ACM, 2011.

[34] Peter Buneman, Adriane Chapman, and James Cheney. Provenance management in curated databases. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 539–550. ACM, 2006.

[35] Peter Buneman, Sanjeev Khanna, and Wang chiew Tan. Why and where: A characterization of data provenance. In *In ICDT*, pages 316–330. Springer, 2001.

[36] Rajkumar Buyya and Srikumar Venugopal. A gentle introduction to grid computing and technologies. *CSI Communications*, 29(1):9–19, July 2005. Computer Society of India (CSI).

[37] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, June 2009.

[38] M. Cafaro and G. Aloisio. *Grids, Clouds and Virtualization*. Computer Communications and Networks. Springer, 2010.

[39] Eddy Caron, Frederic Desprez, and Adrian Muresan. Forecasting for grid and cloud computing on-demand resources based on pattern matching. In *Cloud Computing, Second International Conference, CloudCom 2010*, pages 456–463. IEEE, 2010.

[40] Eddy Caron, Frederic Desprez, and Adrian 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*, CLOUDCOM '10, pages 456–463, Washington, DC, USA, 2010. IEEE Computer Society.

[41] Subhachandra Chandra and Peter M Chen. Whither generic recovery from application faults? a fault study using open-source software. In *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*, pages 97–106. IEEE, 2000.

[42] David Chappell. Introducing the Azure Services Platform. White paper (sponsored by microsoft corporation), DavidChappell & Associates, October 2008.

[43] T.C. Chieu, A. Mohindra, A.A. Karve, and A. Segal. A cloud provisioning system for deploying complex application services. In *e-Business Engineering (ICEBE), 2010 IEEE 7th International Conference on*, pages 125–131, 2010.

[44] Susanta Nanda Tzi-cker Chiueh. A survey on virtualization technologies. 2005.

[45] Tom Coughlin and Mike Alvarado. Angels in our midst: Associative metadata in cloud storage. *Wikibon's summit, November 2nd*, 2010.

[46] Yingwei Cui and Jennifer Widom. Lineage tracing for general data warehouse transformations. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 471–480, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[47] Edward Curry, Desmond Chambers, and Gerard Lyons. Extending Message-Oriented Middleware using Interception. In Antonio Carzaniga and Pascal Fenkam, editors, *Third International Workshop on Distributed Event-Based Systems (DEBS 04) at ICSE 04*, pages 32–37, Edinburgh, Scotland, UK, 2004. IEEE Computer Society.

[48] Tom Austin David W. Cearley David Mitchell Smith Daryl C. Plummer, Thomas J. Bittman. Cloud computing: Defining and describing an emerging phenomenon, 2008.

[49] Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. Elastras: an elastic transactional data store in the cloud. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, HotCloud'09, Berkeley, CA, USA, 2009. USENIX Association.

[50] Partha Dasgupta, Vijay Karamcheti, and Zvi M. Kedem. Transparent distribution middleware for general purpose computations. In *In Proc. of Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 99*, 1999.

[51] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: The montage example, 2008.

[52] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 2011.

[53] D. Dossot and J. Emic. *Mule in Action*. Manning Pubs Co Series. Manning Publications Company, 2009.

[54] D. Dossot and J. Emic. *Mule in Action*, chapter Discovering Mule, pages 03–20. Manning Publications Company, 2009.

[55] D. Dossot and J. Emic. *Mule in Action*, chapter Working with Compnents, pages 139–163. Manning Publications Company, 2009.

[56] D. Dossot and J. Emic. *Mule in Action*, chapter Routing Data With Mule, pages 82–107. Manning Publications Company, 2009.

[57] D. Dossot and J. Emic. *Mule in Action*, chapter Using the Mule API, pages 299–326. Manning Publications Company, 2009.

[58] Idilio Drago, Marco Mellia, Maurizio M. Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside dropbox: understanding personal cloud storage services. IMC '12, pages 481–494, 2012.

[59] Fred Hoch et al. Software as a service: Strategic backgrounder. 2000.

[60] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1, 1999.

[61] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *2008 Grid Computing Environments Workshop*, pages 1–10. IEEE, November 2008.

[62] Apache Software Foundation. Apache axis2/java - next generation web services. Website `http://ws.apache.org/axis2/`, July 2009.

[63] Ritu Garg and Awadhesh Kumar Singh. Fault tolerance in grid computing: State of the art and open issues, feb 2011.

[64] Felix C. Gartner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31, 1999.

[65] The getty research institute. `http://www.getty.edu/research/tools/provenance/index.html`.

[66] M. Greenwood, C. Goble, R. Stevens, J. Zhao, M. Addis, D. Marvin, L. Moreau, and T. Oinn. Provenance of e-Science experiments-experience from bioinformatics. *Proceedings of the UK OST e-Science second All Hands Meeting*, 4, 2003.

[67] Irfan Habib. Virtualization with kvm. *Linux J.*, 2008(166), February 2008.

[68] 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*, 2010 IEEE GLOBECOM Workshops, pages 1345–1350, USA, December 2010. IEEE Communications Society.

[69] Christina Hoffa, Gaurang Mehta, Tim Freeman, Ewa Deelman, Kate Keahey, Bruce Berriman, and John Good. On the use of cloud computing for scientific workflows. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, ESCIENCE '08, pages 640–645, Washington, DC, USA, 2008. IEEE Computer Society.

[70] Soonwook Hwang and Carl Kesselman. Grid workflow: a flexible failure handling framework for the grid. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, 2003*, pages 126–137. IEEE, June 2003.

[71] Ivan Janciak, Peter Brezany, and Fakhri Alam Khan. Workflow 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*. IEEE Computer Society, May 2010.

[72] Jiang Ji-chen and Gao Ming. Enterprise Service Bus and an Open Source Implementation. pages 926–930, September 2007.

[73] Simon Kelly, Karl Heaton, and Jurian Hoogewerff. Tracing the geographical origin of food: The application of multi-element and multi-isotope analysis. *Trends in Food Science and Technology*, 16(12), 2005.

[74] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. *Aspect-oriented programming*. Springer, 1997.

[75] Katherine Kiel and Katherine Tedesco. Stealing history: How does provenance affect the price of antiquities? Working Papers 1105, College of the Holy Cross, Department of Economics, 2011.

[76] Tamas Kifor, László Zsolt Varga, Javier Vazquez-Salceda, Sergio Alvarez, Steven Willmott, Simon Miles, and Luc Moreau. Provenance in agent-mediated healthcare systems. *Intelligent Systems, IEEE*, 21(6):38–46, 2006.

[77] Hwanju Kim, Hyeontaek Lim, Jinkyu Jeong, Heeseung Jo, and Joonwon Lee. Task-aware virtual machine scheduling for I/O performance. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 101–110, New York, NY, USA, 2009. ACM.

[78] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall, 1 edition, November 2004.

[79] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems. *Software Practice and Experience*, 32:135–164, 2002.

[80] Ingolf H. Krueger, Michael Meisinger, Massimiliano Menarini, and Stephen Pasco. Rapid systems of systems integration - combining an architecture-centric approach with enterprise service bus infrastructure. In *In Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI*, 2006.

[81] David P. Lanter. Design of a Lineage-Based Meta-Data Base for GIS. *Cartography and Geographic Information Science*, 18:255–261, 1991.

[82] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas 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*, CLOUD '09, pages 23–31, Washington, DC, USA, 2009. IEEE Computer Society.

[83] Andrew Leung, Minglong Shao, Timothy Bisson, Shankar Pasupathy, and Ethan L. Miller. Spyglass: Fast, scalable metadata search for large-scale storage systems. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST '09)*, February 2009.

[84] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. Nist cloud computing reference architecture. *NIST special publication*, 500:292, 2011.

[85] X. Liu, D. Yuan, D. Cao, G. Zhang, W. Li, J. Chen, Q. He, and Y. Yang. *The Design of Cloud Workflow Systems*. SpringerBriefs in Computer Science. Springer, 2011.

[86] Zhen Liu, Hongbin Huang, Su Deng, and Xueshan Luo. An algorithm for resource discovery based on metadata semantic matching in semantic grid environment. In *Semantics, Knowledge and Grid, 2005. SKG '05. First International Conference on*, pages 58–58, 2005.

[87] Clifford A. Lynch. When documents deceive: trust and provenance as new factors for information retrieval in a tangled web. *J. Am. Soc. Inf. Sci. Technol.*, 52(1):12–17, January 2001.

[88] Peter Macko, Marc Chiarini, and Margo Seltzer. Collecting provenance via the xen hypervisor. In *Workshop on the Theory and Practice of Provenance*, 2011.

[89] Daniel W. Margo and Margo Seltzer. The case for browser provenance. In *First workshop on on Theory and practice of provenance*, TAPP'09, pages 9:1–9:5, Berkeley, CA, USA, 2009. USENIX Association.

[90] Daniel W. Margo and Margo I. Seltzer. The case for browser provenance. In *Workshop on the Theory and Practice of Provenance*, 2009.

[91] Anderson Marinho, Leonardo Murta, Cláudia Werner, Vanessa Braganholo, Sérgio Manuel Serra da Cruz, Eduardo S. Ogasawara, and Marta Mattoso. Provmanager: a provenance management system for scientific workflows. *Concurr. Comput. : Pract. Exper.*, 24(13):1513–1530, September 2012.

[92] Peter Mell and Tim Grance. The NIST Definition of Cloud Computing. Technical report, July 2009.

[93] Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. The requirements of recording and using provenance in e-Science experiments. Technical report, University of Southampton, 2005.

[94] Simon Miles, Sylvia C. Wong, Weijian Fang, Paul Groth, Klaus peter Zauner, and Luc Moreau. Provenance-based validation of e-science experiments. In *In ISWC*, pages 801–815. Springer-Verlag, 2005.

[95] Kiran-Kumar Muniswamy-Reddy, Uri Braun, David A. Holland, Peter Macko, Diana Maclean, Daniel Margo, Margo Seltzer, and Robin Smogor. Layering in provenance systems. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX'09, pages 10–10, Berkeley, CA, USA, 2009. USENIX Association.

[96] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. Provenance-aware storage systems. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 4–4, Berkeley, CA, USA, 2006. USENIX Association.

[97] Kiran-Kumar Muniswamy-Reddy, Peter Macko, and Margo Seltzer. Making a cloud provenance-aware. In *First workshop on on Theory and practice of provenance*, TAPP'09, pages 12:1–12:10, Berkeley, CA, USA, 2009. USENIX Association.

[98] Kiran-Kumar Muniswamy-Reddy, Peter Macko, and Margo Seltzer. Provenance for the cloud. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, pages 15–14, Berkeley, CA, USA, 2010. USENIX Association.

[99] Kiran-Kumar Muniswamy-Reddy and Margo Seltzer. Provenance as first class cloud data. *SIGOPS Oper. Syst. Rev.*, 43(4):11–16, January 2010.

[100] Arun Babu Nagarajan, Frank Mueller, Christian Engelmann, and Stephen L Scott. Proactive fault tolerance for hpc with xen virtualization. In *Proceedings of the 21st annual international conference on Supercomputing*, pages 23–32. ACM, 2007.

[101] Nimbus. `http://www.nimbusproject.org/`. retrieved: Aug 2nd, 2013.

[102] Nitu. Configurability in saas (software as a service) applications. In *Proceedings of the 2nd India software engineering conference*, ISEC '09, pages 19–26, New York, NY, USA, 2009. ACM.

[103] Mohamad Izuddin Bin Nordin and Mahamat Issa Hassan. Cloud resource broker in the optimization of medical image retrieval system: A proposed goal-based request in medical application. In *National Postgraduate Conference (NPC), 2011*, pages 1–5. IEEE, 2011.

[104] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 805–814, New York, NY, USA, 2008. ACM.

[105] J. Pruyne and Hewlett-Packard Laboratories. *Enabling QoS Via Interception in Middleware.* HP Laboratories technical report. Hewlett-Packard Laboratories, 2000.

[106] Shrija Rajbhandari, Ian Wootten, Ali Shaikh Ali, and Omer F. Rana. Evaluating provenance-based trust for scientific workflows. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '06, pages 365–372, Washington, DC, USA, 2006. IEEE Computer Society.

[107] Muralikrishnan Ramane and Bharath Elangovan. A metadata verification scheme for data auditing in cloud environment. *International Journal on Cloud Computing: Services and Architectur*, Aug 2012.

[108] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, and Kun Wang. A distributed self-learning approach for elastic provisioning of virtualized cloud resources. MASCOTS '11, pages 45–54, Washington, DC, USA, 2011. IEEE Computer Society.

[109] Mohamed Amin Sakka, Bruno Defude, and Jorge Tellez. Document provenance in the cloud: constraints and challenges. In *Proceedings of the 16th EUNICE/IFIP WG 6.6 conference on Networked services and applications: engineering, control and management*, EUNICE'10, pages 107–117, Berlin, Heidelberg, 2010. Springer-Verlag.

[110] F. Salfner and M. Malek. Reliability modeling of proactive fault handling. Technical Report 209, Department of Computer Science, Humboldt-Universität zu Berlin, Germany, 2005.

[111] C.W.A.C.W. Samsudin. *Data Provenance for E-social Science Cloud Applications*. Final year project–University of Leeds (School of Computing Studies), 2010/2011. University of Leeds, School of Computing Studies, 2011.

[112] Can Sar and Pei Cao. Lineage file system. Online at http://crypto.stanford.edu/ cao/lineage.html, January 2005.

[113] Richard E Schantz and Douglas C Schmidt. Middleware for distributed systems: Evolving the common structure for network-centric applications. 2002.

[114] Maren Scheffel, Katja Niemann, Abelardo Pardo, Derick Leony, Martin Friedrich, Kerstin Schmidt, Martin Wolpers, and Carlos Delgado Kloos. Usage pattern recognition in student activities. In *Proceedings of the 6th European conference on Technology enhanced learning: towards ubiquitous learning*, EC-TEL'11, pages 341–355, Berlin, Heidelberg, 2011. Springer-Verlag.

[115] Carlos Scheidegger, David Koop, Emanuele Santos, Huy Vo, Steven Callahan, Juliana Freire, and Cláudio Silva. Tackling the provenance challenge one layer at a time. *Concurr. Comput. : Pract. Exper.*, 20(5):473–483, April 2008.

[116] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A Survey of Data Provenance Techniques. Technical report, Computer Science Department, Indiana University, Bloomington IN, 2005.

[117] Yogesh L Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *ACM Sigmod Record*, 34(3):31–36, 2005.

[118] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A framework for collecting provenance in data-centric scientific workflows. In *ICWS*, pages 427–436, 2006.

[119] Yogesh L. Simmhan, Beth Plale, Dennis Gannon, and Suresh Marru. Performance evaluation of the karma provenance framework for scientific workflows. In *IPAW*, pages 222–236. Springer, 2006.

[120] Michael Smit, Przemyslaw Pawluk, Bradley Simmons, and Marin Litoiu. A web service for cloud metadata. In *Proceedings of the 2012 IEEE Eighth World Congress on Services*, SERVICES '12, pages 361–368, Washington, DC, USA, 2012. IEEE Computer Society.

[121] Diomidis Spinellis. Another level of indirection. In Andy Oram and Greg Wilson, editors, *Beautiful Code: Leading Programmers Explain How They Think*, chapter 17, pages 279–291. O'Reilly and Associates, Sebastopol, CA, 2007.

[122] Robert Stevens, Kevin Glover, Chris Greenhalgh, Claire Jennings, Simon Pearce, Peter Li, Melena Radenkovic, and Anil Wipat. Performing in silico Experiments on the Grid: A Users Perspective. In *Proceedings UK e-Science programme All Hands Meeting*, pages 43–50, 2003.

[123] Martin Szomszor and Luc Moreau. Recording and reasoning over data provenance in web and grid services. In Robert Meersman, Zahir Tari, and Douglas C. Schmidt, editors, *CoopIS/-DOA/ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 603–620. Springer, 2003.

[124] Martin Szomszor and Luc Moreau. Recording and reasoning over data provenance in web and grid services. In *In Int. Conf. on Ontologies, Databases and Applications of Semantics, volume 2888 of LNCS*, pages 603–620, 2003.

[125] Jian Tan, Parijat Dube, Xiaoqiao Meng, and Li Zhang. Exploiting resource usage patterns for better utilization prediction. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems Workshops*, ICDCSW '11, pages 14–19, Washington, DC, USA, 2011. IEEE Computer Society.

[126] Wang Chiew Tan. Provenance in databases: Past, current, and future. volume 30, pages 3–12, 2007.

[127] Val Tannen. Provenance for database transformations. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 1–1, New York, NY, USA, 2010. ACM.

[128] R. H. Tykot. Scientific methods and applications to archaeological provenance studies. In *Proceedings of the International School of Physics: IOS Press, Amsterdam 2004*, pages 407–432, 2004.

[129] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, December 2008.

[130] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th Annual Southeast Regional Conference*, ACM SE '10, pages 42:1–42:6, New York, NY, USA, 2010. ACM.

[131] Jens-Sönke Vöckler, Gideon Juve, Ewa Deelman, Mats Rynge, and Bruce Berriman. Experiences using cloud computing for a scientific workflow application. pages 15–24, USA, 2011. ACM.

[132] L. Wall and A. Lader. *Building Web services and .NET applications*. Application Development Series. McGraw-Hill/Osborne, 2002.

[133] Simon Wardley, Etienne Goyer, and Nick Barcet. Ubuntu Enterprise Cloud Architecture. *Technical White Paper*, August 2009.

[134] Craig D. Weissman and Steve Bobrowski. The design of the force.com multitenant internet application development platform. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 889–896, New York, NY, USA, 2009. ACM.

[135] Ying Yang, Xindong Wu, and Xingquan Zhu. Combining proactive and reactive predictions for data streams. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 710–715, New York, NY, USA, 2005. ACM.

[136] Jing Zhang, Adriane Chapman, and Kristen Lefevre. Do you know where your data's been? — tamper-evident database provenance. In *Proceedings of the 6th VLDB Workshop on Secure Data Management*, SDM '09, pages 17–32, Berlin, Heidelberg, 2009. Springer-Verlag.

[137] Olive Qing Zhang, Markus Kirchberg, Ryan K. L. Ko, and Bu Sung Lee. How to track your data: The case for cloud computing provenance. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, CLOUDCOM '11, pages 446–453, Washington, DC, USA, 2011. IEEE Computer Society.

[138] Jun Zhao, Chris Wroe, Carole Goble, Robert Stevens, Dennis Quan, and Mark Greenwood. Using Semantic Web Technologies for Representing E-science Provenance. In *The Semantic Web ISWC 2004*, pages 92–106. 2004.