Scientific Workflow Execution Reproducibility using Cloud-Aware Provenance

Mian Khawar Hasham Ahmad

The University of the West of England
Faculty of Environment and Technology

This thesis is submitted for the degree of
Doctor of Philosophy

March 2016

Abstract

Scientific experiments and projects such as CMS and neuGRIDforYou (N4U) are annually producing data of the order of Peta-Bytes. They adopt scientific workflows to analyse this large amount of data in order to extract meaningful information. These workflows are executed over distributed resources, both compute and storage in nature, provided by the Grid and recently by the Cloud. The Cloud is becoming the playing field for scientists as it provides scalability and on-demand resource provisioning. Reproducing a workflow execution to verify results is vital for scientists and have proven to be a challenge. As per a study (Belhajjame et al. 2012) around 80% of workflows cannot be reproduced, and 12% of them are due to the lack of information about the execution environment. The dynamic and on-demand provisioning capability of the Cloud makes this more challenging. To overcome these challenges, this research aims to investigate how to capture the execution provenance of a scientific workflow along with the resources used to execute the workflow in a Cloud infrastructure. This information will then enable a scientist to reproduce workflow-based scientific experiments on the Cloud infrastructure by re-provisioning the similar resources on the Cloud.

Provenance has been recognised as information that helps in debugging, verifying and reproducing a scientific workflow execution. Recent adoption of Cloud-based scientific workflows presents an opportunity to investigate the suitability of existing approaches or to propose new approaches to collect provenance information from the Cloud and to utilize it for workflow reproducibility on the Cloud. From literature analysis, it was found that the existing approaches for Grid or Cloud do not provide detailed resource information and also do not present an automatic provenance capturing approach for the Cloud environment. To mitigate the challenges and fulfil the knowledge gap, a provenance based approach, ReCAP, has been proposed in this thesis. In ReCAP, workflow execution reproducibility is achieved by (a) capturing the Cloud-aware provenance (CAP), b) re-provisioning similar resources on the Cloud and re-executing the workflow on them and (c) by comparing the provenance graph structure including the Cloud resource information, and outputs of workflows. ReCAP captures the Cloud resource information and links it with the workflow provenance to generate Cloud-aware provenance. The Cloud-aware provenance consists of configuration parameters relating to hardware and software describing a resource on the Cloud. This information once captured aids in re-provisioning the same execution infrastructure on the Cloud for workflow re-execution. Since resources on the Cloud can be used in static or dynamic (i.e. destroyed when a task is finished) manner, this presents a challenge for the devised provenance capturing approach. In order to deal with these scenarios, different capturing and mapping approaches have been presented in this thesis. These mapping approaches work outside the virtual machine and collect resource information from the Cloud middleware, thus they do not affect job performance. The impact of the collected Cloud resource information on the job as well as on the workflow execution has been evaluated through various experiments in this thesis. In ReCAP, the workflow reproducibility is

verified by comparing the provenance graph structure, infrastructure details and the output produced by the workflows. To compare the provenance graphs, the captured provenance information including infrastructure details is translated to a graph model. These graphs of original execution and the reproduced execution are then compared in order to analyse their similarity. In this regard, two comparison approaches have been presented that can produce a qualitative analysis as well as quantitative analysis about the graph structure. The ReCAP framework and its constituent components are evaluated using different scientific workflows such as ReconAll and Montage from the domains of neuroscience (i.e. N4U) and astronomy respectively. The results have shown that ReCAP has been able to capture the Cloud-aware provenance and demonstrate the workflow execution reproducibility by re-provisioning the same resources on the Cloud. The results have also demonstrated that the provenance comparison approaches can determine the similarity between the two given provenance graphs. The results of workflow output comparison have shown that this approach is suitable to compare the outputs of scientific workflows, especially for deterministic workflows.

This thesis is dedicated to my loved ones who departed this world during my studies.

My mother, whose love and memories I will cherish forever.

Acknowledgements

As is the case with every journey, this PhD journey was not possible without the support and love I received from my supervisory team, family and colleagues. I am thankful to Professor Richard McClatchey, my PhD supervisor, who has been supportive in the funding arrangements and the university administrative procedures throughout this research work. Due to his vast experience in research, his valuable suggestions helped me to pull myself out of some difficult situations.

I really feel indebted to Dr Kamran Munir, my PhD co-supervisor, for being tolerant, and always helpful as a colleague and supervisor. His guidance in general and technical suggestions in particular have really guided me during this study. His calm and friendly demeanour allowed me to discuss and share ideas with him at any time. I am thankful to Dr Jetendr Shamdasani, who joined my supervision team in later half of my PhD but provided friendly and helpful reviews and suggestions. Apart from my current supervision team, I would also like to appreciate the efforts and support provided by Dr Saad Liaquat, who was part of my supervision team before moving to Pakistan.

Without the support of parents and family, this long journey of four years would not have been possible. During this journey, my mother passed away but her wishes and prayers are still with me. The support I received from my father cannot be described in words. My wife Rumaisa, whose patience and care allow me to complete this work. I am eternally grateful to her for sharing my burden.

Contents

List of Figures

List of Tables

Listings

List of Algorithms

Acronyms

APIs Application Programming Interfaces. 18

CAP Cloud-aware Provenance. 14, 51, 95

CRM Cloud Resource Manager. vi, 60

CSM Cloud Storage Manager. 60

DAG Directed Acyclic Graph. 12, 23, 56

DAGMan DAG Manager. 24, 56

DAGs Directed Acyclic Graphs. 45

HTTP HyperText Transfer Protocol. 51, 55

IaaS Infrastructure-as-a-Service. 14, 22

JSON JavaScript Object Notation. 56, 63

LRM Local Resource Manager. 17

MIPS Million Instructions per Second. 42, 58

mIR myGrid Information Repository. 22

OSDC Open Science Data Cloud. 89, 91, 94

PaaS Platform-as-a-Service. 14

PM Physical Machine. 30

PRECIP Pegasus Repeatable Experiments in the Cloud. 36, 97

RDF Resource Description Framework. 22

SaaS Software-as-a-Service. 14

SSH Secure Shell. 28

URL Universal Resource Locator. 17

VDC Virtual Data Catalogue. 21

VM Virtual Machine. 6, 16, 27, 51

WMS Workflow Management System. 44, 50, 51

WS WrapperService. 55, 58

Chapter 1

Introduction

> An experiment is reproducible until another laboratory tries to repeat it.
>
> *Alexander Kohn*

In this chapter, the problem domain is defined and the motivation behind this research study is provided. The research hypothesis, research questions and the research methodology that has been used to investigate the hypothesis in this thesis are also presented. At the end of this chapter, a structure of this thesis is outlined. Finally, a list of papers produced as outcome of this research is also given at the end.

1.1 Background

The scientific community is experiencing a data deluge due to the generation of large amounts of data in modern scientific experiments that include projects such as the Laser Interferometer Gravitational Wave Observatory (LIGO) (Abramovici et al. 1992), the Large Hadron Collider (LHC)[1], and projects such as neuGRID (Munir et al. 2013). In particular, the neuGRID community is utilising scientific workflows to orchestrate the complex processing of its data analysis. A large pool of compute and data resources are required to process this data, which has been available through the Grid (Foster and Kesselman 1999) and is now also being offered by Cloud-based infrastructures.

Cloud computing (Mell and Grance 2011) has emerged as a new computing and storage paradigm, which is dynamically scalable and usually works on a pay-as-you-go cost model. It aims to share resources to store data and to host services transparently among users at a massive scale (Mei, Chan and Tse 2008). Its ability to provide an on-demand computing infrastructure with scalability enables

[1] http://lhc.cern.ch

distributed processing of complex scientific workflows for the scientific community (Deelman et al. 2008b). Recent work (Juve and Deelman 2010) is now experimenting with Cloud infrastructures to assess the feasibility of executing workflows on the Cloud.

An important consideration during this data processing is to gather data that can provide detailed information about both the input and the processed output data, and the processes involved to verify and reproduce a workflow execution. Such data are termed as "Provenance" in the scientific literature. Provenance is defined as the derivation history of an object (Simmhan, Plale and Gannon 2005). The Oxford dictionary defines provenance as "the place of origin or earliest known history of something" (Oxford 2015) as provenance would initially be used to provide chronological information about the ownership or location of an object. Different terms such as lineage and pedigree are also used in literature to describe the same concept, however, the term provenance is used in this thesis.

The provenance information can be used to debug and verify the execution of a workflow, to aid in error tracking and reproducibility (Freire et al. 2008). This is of vital importance for scientists in order to make their experiments verifiable and reproducible. The collection and use of provenance information enables them to iterate on the scientific method, to evaluate the process and results of other experiments and to share their own experiments with other scientists (Azarnoosh et al. 2013). Thus, provenance information can be considered as a logbook that captures all the steps that were involved in the actual derivation of the result and also can be used to replay or reproduce the execution that lead to that result for experiment verification (Moreau 2010). The execution of scientific workflows in Clouds, which provide dynamic and on-demand resources, brings to the fore the need to collect provenance information that is necessary to ensure the reproducibility of workflow-based experiments on the Cloud.

1.2 Motivation

A research study (Belhajjame et al. 2012) conducted to evaluate the reproducibility of scientific workflows has shown that around 80% of the workflows cannot be reproduced, and 12% of them are due to the lack of information about the execution environment (Santana-Pérez and Pérez-Hernández 2014). This information affects a workflow execution in multiple ways. A workflow execution can not be reproduced if the underlying execution environment does not provide the libraries (i.e. software dependencies) that are required for workflow execution. Besides the software dependencies, hardware dependencies related to an execution environment can also affect a workflow execution. It can affect a workflow's overall execution performance and also job failure rate. This effect on the experiment performance has also been highlighted by Kanwal et al. (2015). For instance, a data-intensive job can perform better with 2 GB of RAM because it can accommodate more data in RAM, which is a faster medium than hard disk. Conversely, the job's performance will degrade if a resource of 1 GB RAM is allocated to this job as less data can be placed in RAM. Moreover, it is also

possible that jobs will remain in waiting queues or fail during execution if their required hardware dependencies are not met. This becomes more of a challenging issue in the context of the Cloud where resources can be created or destroyed at runtime.

Cloud computing presents a dynamic environment in which resources are provisioned on-demand. For this, a user submits resource configuration information as a resource provision request to the Cloud infrastructure. A resource is allocated to the user if the Cloud infrastructure can meet the submitted resource configuration requirements. Moreover, the pay-as-you-go model in the Cloud puts constraints on the lifetime of a Cloud resource. For instance, one can acquire a resource for a lifetime but one has to pay accordingly for the time used. This means that a resource is released once a task is finished or payment has ceased. In order to acquire the same resource, one needs to know the configuration of that old resource. This is exactly the situation with reproducing a workflow experiment on the Cloud. In order to reproduce a workflow execution, a researcher must know the resource configurations used earlier in the Cloud. This enables him to re-provision similar resources and reproduce the workflow execution.

The dynamic and geographically distributed nature of Cloud computing makes the capturing and processing of provenance information a major research challenge (Zhao et al. 2011; Vouk 2008). Contrary to Grid computing, the resources in Cloud computing are virtualised and provisioned on-demand, and released when a task is complete (Foster et al. 2008). Generally, an execution in Cloud based environments occurs transparently to the scientist, i.e. the Cloud infrastructure behaves like a black box. Therefore, it is critical for scientists to know the parameters that have been used and what data products were generated in each execution of a given workflow (SMS et al. 2011; Shamdasani, Branson and McClatchey 2012). Due to the dynamic nature of the Cloud the exact resource configuration should be known in order to reproduce the execution environment. Due to these reasons, there is a need to capture information about the Cloud infrastructure along with workflow provenance, to aid in the repeatability of experiments.

1.2.1 N4U: Case Study

The N4U[2], an EU FP7 project, is a follow-on project to the neuGRID that was aimed to help neuroscientists carry out high-throughput imaging research for individual patient diagnosis especially for Alzheimer's disease. Using its infrastructure in 2009, it was possible to extract an Alzheimer's disease biomarker (3D cortical thickness with Freesurfer and CIVET) from 6,500 MR scans within 2 weeks whereas the process would have previously taken five years using past computational imaging analysis techniques. To achieve high-throughput imaging analysis of complex design, the N4U uses the concept of pipelines, which are analogous to scientific workflows. These workflows are executed using CRISTAL (Branson et al. 2012) on the N4U Grid infrastructure. The requirements analysis of

[2]https://neugrid4you.eu/

N4U (Spulber, Damangir and Wahlund 2013; Spulber, Damangir and Wahlund 2012) shows that users are interested in capturing provenance information related to the executed workflows. The key provenance requirement is to capture provenance data related to workflow execution so that a workflow execution can be validated. This workflow provenance information includes storing workflow and input parameters and information on intermediary execution steps. It also requires to store output and error messages produced as a result of a workflow execution. This shows the key interest of N4U users in workflow provenance and also using this information to verify and validate a workflow execution. At present, the collected provenance information in N4U does not provide information about the execution infrastructure. Since this research work is based on the Cloud environment, these N4U requirements become more challenging given the nature of the Cloud discussed above. Furthermore, the keen interest of the N4U users in the workflow validation using workflow provenance, means it is important to devise a mechanism that helps researchers in reproducing a workflow execution. Therefore, this research study aims to capture Cloud-aware provenance information for workflows executing on the Cloud to provide support for workflow execution reproducibility on the Cloud.

1.3 Research Problem Area

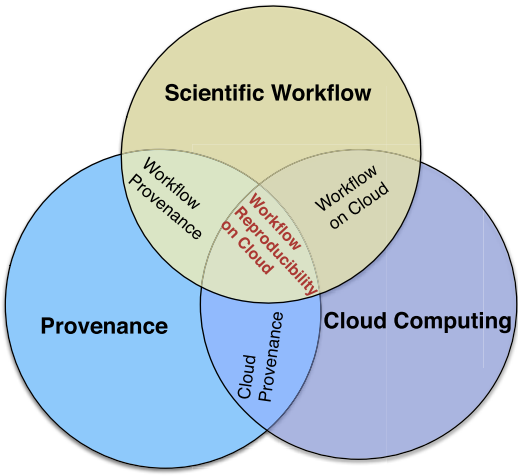


Figure 1.1: Venn Diagram highlighting the areas this research study is focusing on; red font indicates the research aim

Workflow provenance provides data that can assist in achieving workflow reproducibility because it can provide information about all the processes and data used during execution. As discussed earlier, the workflow reproducibility is affected by the lack of information about the underlying execution environment (hardware and software) (Belhajjame et al. 2012). This problem becomes more challenging in the case of the Cloud since it offers more dynamic resource provisioning. On the Cloud, required resources are provisioned by sending resource requests to the Cloud IaaS (Infrastructure as a Service) layer. The Cloud IaaS layer provides virtualized resources on top of the actual

physical resources. The resource request contains information about the resource configurations that are demanded from the Cloud. Based on this, the Cloud IaaS layer creates resources for a user. Moreover, in dynamic situations, the Cloud resources are released (destroyed) once a job is finished. Due to this, it is necessary to devise a mechanism that can enable the capture of the Cloud resource information along with workflow provenance to achieve workflow reproducibility (as shown in Figure 1.2). Consequently, this research study will focus on three broad domains i.e. Cloud computing, scientific workflow and its reproducibility using provenance (as shown in Figure 1.1).

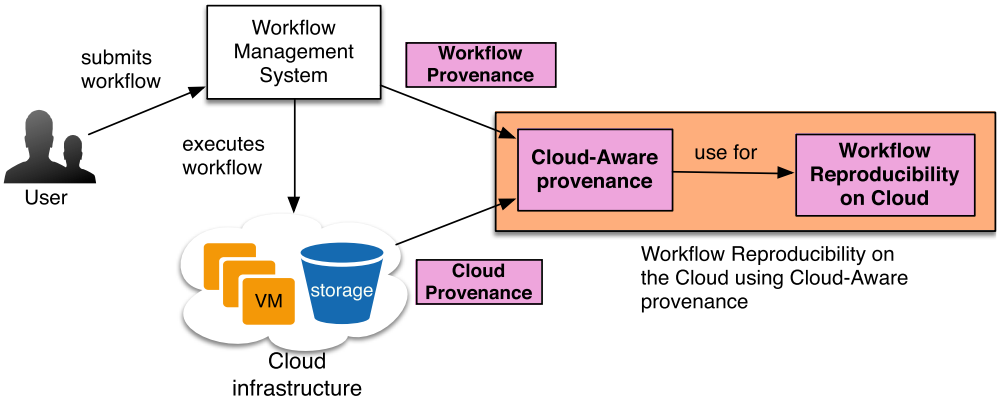


Figure 1.2: Illustrating the focus of this research study; pink background: research domains, orange background: research aim

1.4 Hypothesis and Research Questions

This section provides the hypothesis that will drive this research study. Its related research questions are also listed and discussed in this section.

"A Cloud-Aware provenance capturing approach can be used to collect provenance of the execution concerning Cloud-based scientific workflows to demonstrate the reproducibility of the workflow-based scientific experiments."

In order to evaluate this hypothesis, the following research questions will be answered.

1. **To what extent are existing provenance capturing techniques adequate for capturing provenance of a Cloud-based scientific workflow?**

   As of now, different systems such as CRISTAL (Branson et al. 2012) and Pegasus (Deelman et al. 2004) exist to capture provenance data from scientific workflows executing on a Grid. However, little work has been done for Cloud-based scientific workflows. Some of workflow execution systems e.g. (Juve and Deelman 2010; Vöckler et al. 2011) use the Cloud and

create a 'Virtual Cluster' (Klinginsmith, Mahoui and Wu 2011) on the Cloud resources, which mimics Grid-based systems, and then capture the provenance information as done on the Grid. However, the Cloud operates on different layers, including application, middleware and virtualization. In this thesis, a workflow management system is placed at the application layer providing the user with set of workflow related operations. The virtual resources i.e. virtual machines and storage used to carry out workflow execution are provided at the infrastructure layer, also called IaaS layer. Due to virtualization, a user does not know where his computing or data resource is hosted and the configurations of the used resources. To provide comprehensive provenance, a Cloud-aware provenance information needs to be captured in a Cloud-based execution environment.

2. **How can provenance information be captured at different layers in a Cloud environment and interlinked?**

   To address this research question, we need to investigate techniques for capturing provenance from different layers in Cloud, where the application layer consists of a workflow management system and the infrastructure layer consists of virtualized resources. Linking this multi-layered or Cloud-aware provenance information is a related research challenge. This will help in aggregating the captured provenance information. This also requires investigating the extent to which workflow provenance capture in different layers of a Cloud can be achieved.

3. **How can correctness and completeness of the collected provenance be measured?**
   The complete provenance for an experiment is captured by a set of objects that can identify all the intermediate steps involved during the experiments (Prat and Madnick 2008). This process begins with an abstract description of the workflow (experiment), before it is bound to specific services, tools, data sets, etc., and continues until the end of the execution of the experiment. Moreover, completeness means that all the required elements are present in the captured provenance. The focus of this research study is on the provenance of a workflow in its execution phase over the Cloud infrastructure. Thus, this research question will investigate the ways to determine the correctness and completeness of the captured workflow provenance on Cloud for its reproducibility purpose. In doing so, it will also evaluate the proposed framework and its components to verify in detail their functioning.

4. **How can we demonstrate workflow reproducibility on the Cloud infrastructure using Cloud-aware provenance?**
   In order to demonstrate the workflow reproducibility on the Cloud infrastructure, it is particularly important to first evaluate that the captured Cloud-aware provenance information is correct and sufficient to re-provision resources on the Cloud. It is also necessary to demonstrate that both the workflow executions i.e. original and reproduced were performed on the same Cloud infrastructure and had produced the same outputs. Once the Cloud-aware provenance information is collected, there should be a mechanism to use it to provision resources

i.e. Virtual Machine (VM) on the Cloud and re-execute the workflow. After re-executing the workflow on the Cloud, there should also be a mechanism to compare the two workflow executions in order to determine the reproducibility. With this research question, this research study will seek a mechanism to demonstrate the reproducibility of a workflow using Cloud-aware provenance information.

1.5 Contribution to Knowledge

The following are the key contributions of this research study.

- **Mapping Workflow Provenance with Cloud infrastructure information:** Given the critical analysis of the state of the art and workflow reproducibility on the Cloud, a workflow reproducibility framework has been identified for reproducing a workflow execution on the Cloud. The key factors of this framework are the conservation of hardware and software resources used to execute a workflow on the Cloud. In order to capture the Cloud resource information, different mapping strategies have been devised for two different environments i.e. the Static and Dynamic in the Cloud. In this regard, an automated and '*outside the Cloud*' approach has been devised to retrieve the configuration of a Cloud resource which was used to execute a job of a workflow. In this approach, no additional job or process needs to be running within a virtual machine. The contributions in this category are listed below. Two use cases of Cloud environments i.e. static and dynamic environment, have been identified and different mapping mechanisms have been designed to mitigate the challenges of these use cases.

    1. **Mapping approaches in Static Cloud environment:** One approach has been presented that captures Cloud-aware provenance in the Static Cloud environment, in which resources remain available after a workflow has completed.
    2. **Mapping approaches in Dynamic Cloud environment:** Two approaches have been presented that aid in capturing Cloud-aware provenance in the dynamic Cloud environment, in which resources may not be accessible once a workflow has finished.

- **Workflow Provenance comparison:** The collected Cloud-aware provenance is expressed as a directed graph. This graph also includes the Cloud resource and output information. To best of our knowledge, the addition of execution resources as part of a provenance graph has not been hitherto that can be used for this purpose. The graph of the original execution is compared with the reproduced workflow provenance and the algorithm determines the similarities between the two given provenance graphs. This comparison also enables a user to investigate the difference (if any) between two given provenance graphs. The proposed algorithm takes into account various factors, which have been discussed in Section 4.2. The contributions in this category are given below.

1. **Provenance Graph Comparison:** In this thesis, two graph comparison approaches have been presented that take cloud infrastructure into account while performing the analysis. One algorithm only provides qualitative analysis by providing results in either True (for complete match) or False (for mismatch). In order to determine quantitatively, a numerical comparison approach has also been presented that provides quantitative graph comparison i.e. the degree of similarity.
2. **Workflow Output Comparison:** Since outputs produced by the workflows are also part of the comparison, an approach has been presented in this thesis to compare and verify the reproduced outputs. The results have shown that this approach is applicable for different workflows such as Montage. It can also work with the N4U workflow i.e. ReconAll by including the files related to the brain analysis.
3. **Provenance Correctness and Completeness Analysis:** Two approaches have been presented in this thesis that verifies the completeness of the captured Cloud-aware provenance. This analysis facilitates trust in the captured provenance by ensuring ReCAP has collected the Cloud infrastructure information correctly for all the jobs in the workflow.

- **Workflow based experiment reproducibility on the Cloud:** A framework has been presented that is used to re-provision similar resources on the Cloud using the Cloud-aware provenance information captured by the proposed system. After re-provisioning the resources, the workflow is re-executed and its provenance is captured and compared with the original workflow.

1.6 Research Methodology

As this research deals with a practical problem which requires investigation into existing workflow provenance and reproducibility approaches, particularly, on Cloud, the research methodology utilised in undertaking this research is a combination of scientific experiment i.e. quantitative and qualitative nature (McCusker and Gunaydin 2015). This means that a subset of the research problem is investigated through prototype and its experiments while another subset is analysed through literature review. In order to understand the practical requirements in real world scenario, a case study method (Myers 1997) of a qualitative approach (O'Brien 1998) has been adopted. The case study method helps in understanding the existing application of workflow in scientific domain and also user requirements towards workflow provenance and reproducibility. This also provides sample workflows which are later used in the prototype evaluation phase. In order to evaluate the proposed approaches, a prototype is implemented and various aspects of the architecture are empirically analysed through experiments.

Since research prototype design is not a linear process, the adopted research methodology is both linear as well as iterative, as shown in Figure 1.3. In the *Observation* phase, the background

and the motivation of the research problem is identified. This step is informed by a literature review leading to the formation of the hypothesis and associated research questions. Based on the identified hypothesis and questions, a literature review has been performed to understand the specifics of the problem and also the related work that has been carried out in the domain. The analysis of the literature helps in understanding what has been achieved and what is still missing in the context of this research study and research problem. Moreover, the analysis of the literature (discussed in Chapter 2) also helps in answering **Research Question 1** i.e. *To what extent are existing provenance capturing techniques adequate for capturing provenance of a Cloud-based scientific workflow?*

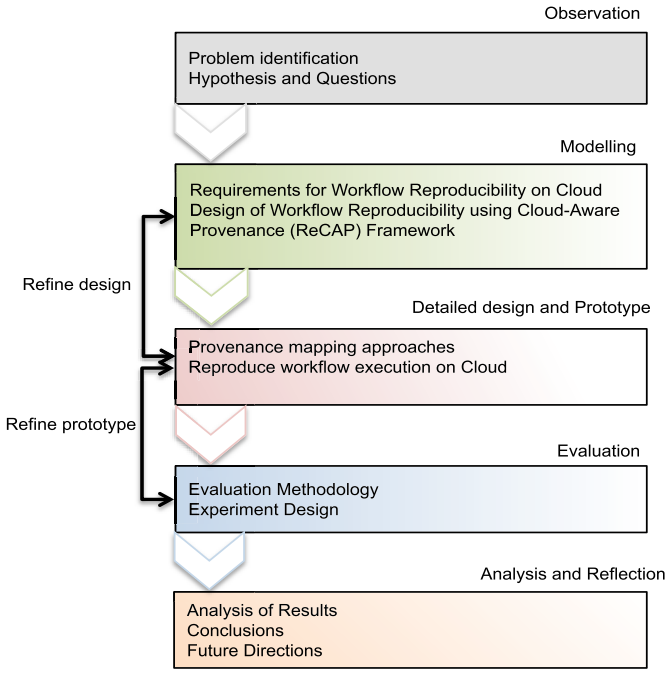


Figure 1.3: Research Phases

In the *Modelling* phase, based on the literature review, a workflow reproducibility model i.e. a set of requirements for the Cloud environment have been derived. These requirements aid in understanding what factors are important in achieving reproducibility especially on a Cloud infrastructure. Moreover, a case study from N4U is presented to aid in understanding how workflows are executed and to discover the requirements for reproducibility. This case study is then analysed in the context of the Cloud and the requirements that may arise for the Cloud environment. This information serves the basis for the design of the proposed ReCAP framework. The provenance data and its correctness and completeness have been analysed through the literature analysis (discussed in Section 3.4). This aids in answering **Research Question 3** i.e. *How can correctness and completeness of the collected provenance be measured?* The outcome of this phase is the proposed model of the ReCAP framework that tends to fulfil the requirements discovered from analysing the N4U domain, Cloud computing, and state of the art systems.

In the *Detailed design and Prototype* phase, the ReCAP framework is refined by analysing differ-

ent Cloud usage models that have resulted in different job-to-Cloud resource mapping algorithms, which addresses **Research Question 2** i.e. *How can provenance information be captured at different layers in a Cloud environment and interlinked?* For implementing and testing the proposed system, a Cloud infrastructure is needed to execute scientific workflows and to capture their provenance. In this study, Pegasus, a workflow management system, has been selected and justified in Section 3.5.2. Pegasus has been used in various studies (e.g. Vöckler et al. 2011; Juve et al. 2009) to execute workflows on Cloud infrastructures. In this research study, it has been used to execute workflows on the configured Cloud infrastructure.

In the *Evaluation* phase, a set of experiments to test the suitability of the proposed model against the elements of the hypothesis has been identified. In doing so, a detailed evaluation strategy has been devised (presented in Chapter 5) that evaluates different facets of the proposed system. To demonstrate the workflow reproducibility, Cloud-aware provenance is linked with workflow provenance using the proposed mapping approaches (discussed in Section 4.1). This interlinked provenance information has been used to provision similar virtual machine on a Cloud infrastructure in order to provide a similar execution infrastructure for the workflow re-execution. Once it has been achieved, a workflow execution is reproduced on the Cloud using the proposed approach (discussed in Section 4.3). By analysing the provenance graphs, at different levels such as structure, infrastructure and output, workflow execution reproducibility is demonstrated. This aids in answering **Research Question 4** i.e. *How can we demonstrate workflow reproducibility on the Cloud infrastructure using Cloud-aware provenance?*

In the *Analysis and Reflection* phase, the results generated from the experiments are analysed to evaluate the proposed framework. These results are investigated to verify the theoretical understanding of the proposed model. Based on the results, conclusions are drawn which eventually help in answering the main hypothesis, which drives this research. In this phase, the proposed framework is critically analysed to verify the key contributions. The overall approach is reflected upon to identify shortcomings which will then pave the way for future research directions.

1.7 Thesis Structure

- Chapter 2 - Literature Survey

  In order to analyse the relevant literature for this research, this chapter discusses the state of the art in three main areas which have been identified in Section 1.3. These three areas are; (a) Scientific Workflows and Provenance; (b) Workflow Provenance Management in the Grid and the Cloud; and (c) Reproducibility approaches. The analysis of the existing approaches has been performed by focusing on the work done for workflow reproducibility especially in Cloud, which is the aim of this research.

- Chapter 3 - Workflow Reproducibility Requirements and the Proposed Architecture

  This chapter discusses the computational reproducibility as proposed in the literature. It also presents a workflow reproducibility model after acquiring knowledge from existing work and also keeping the Cloud context in mind. This chapter also discusses N4U, under which this research study was conducted, its provenance requirements, especially in Cloud context, it current execution environment and the possibility of using existing technologies being used in N4U. This chapter also discusses and justifies an existing workflow execution model in the Cloud, which has been used in research that has been used in the proposed solution.

- Chapter 4 - Reproducing Workflow using Cloud-Aware Provenance (ReCAP) - Implementation

  It presents and discusses the proposed solution, ReCAP, to capture workflow and Cloud infrastructure layer information and interlink together. It also presents the Cloud resource to workflow job mapping algorithms and how a workflow can be reproduced by re-provisioning the similar resources on the Cloud using the Cloud-aware provenance information.

- Chapter 5 - Evaluation

  This chapter presents a detailed evaluation methodology that has been used to verify this research work. The presented methodology evaluates different aspects of the ReCAP system using different experiments which have been designed for this purpose. A complete evaluation of the ReCAP system has been performed by evaluating the proposed approaches and using the scientific workflows from different domains.

- Chapter 6 - Results and Analysis

  This chapter discusses the results obtained from the experimental results discussed in Chapter 5. The results are analysed and related to the research questions and research hypothesis.

- Chapter 7 - Conclusion and Future work

  In this final chapter the conclusions from this thesis are presented. It answers the research questions in the light of this research work. The major contributions from this research are detailed in relation to the previously presented hypothesis and research questions. This chapter then presents directions for future work.

1.8 Publications

The following are the publications achieved as a result of this research study.

- Hasham K., Munir K., and McClatchey R.(2015). Using Cloud-Aware Provenance to Reproduce Scientific Workflow Execution on Cloud. In Proceedings of the 5th International

Conference on Cloud Computing and Services Science (CLOSER), pp. 49-59, 20 May 2015. DOI: 10.5220/0005452800490059

- Hasham, K., Munir, K., Shamdasani, J., McClatchey, R.(2014). Scientific Workflow Repeatability through Cloud-Aware Provenance. In Utility and Cloud Computing (UCC), IEEE/ACM 7th International Conference on , vol., no., pp.951,956, 8-11 Dec. 2014, doi: 10.1109/UCC.2014.155

- Munir, K., Ahmad, K. H., and McClatchey, R.(2015). Development of a large-scale neuroimages and clinical variables data atlas in the neuGRID4You (N4U) project. Journal of Biomedical Informatics, vol. 57, October 2015, pp. 245-262, http://dx.doi.org/10.1016/j.jbi.2015.08.004

- Munir, K., Kiani, L. S., Hasham, K., McClatchey, R., Branson, A., and Shamdasani, J.(2014). Provision of an Integrated Data Analysis Platform for Computational Neuroscience Experiments. Journal of Systems and Information Technology, vol. 16 No. 3 pp. 150-169 DOI 10.1108/JSIT-01-2014-0004 August 2014. Emerald Publishers.

Chapter 2

Literature Survey

Scientific workflows have been used to orchestrate complex data processing. These workflows are executed in distributed environments such as the Grid or recently the Cloud. In scientific research, result reproducibility is essential as it validates the claim made by the research. Same is applied to the research conducted through executing workflows. In order to achieve this, provenance data has been recognized as a means to provide the audit trail of the execution, thus it aids in achieving reproducibility. Due to the dynamic nature of the Cloud, capturing provenance for workflows executing on the Cloud infrastructure and then using it to reproduce a workflow execution on the Cloud becomes a challenge, which is the driving force behind this research work. Since this research resolves around three main categories (highlighted in Section 1.3): 1) Scientific Workflows, 2) Cloud and 3) Provenance and their combinations i.e. a) Workflow execution on Cloud, b) Workflow provenance management and c) Reproducibility using provenance, this chapter will attempt to review literature in these areas and highlight the knowledge gap. The analysis of existing literature will be conducted by keeping the main focus of this research, i.e. workflow execution reproducibility on the Cloud, in perspective. Figure 2.1 presents an overview of the literature roadmap carried out in this research work.

In this chapter, an introduction to scientific workflows and its lifecycle is described (Section 2.1.1) to explain different phases of a workflow and to highlight the focus of this research i.e. workflow execution provenance. This chapter then discusses two types of provenance data (Section 2.1.2) to further narrow down the provenance information that this research is focusing on. It is then followed by an introduction to the Cloud or Cloud computing, which is now being explored as a new computing platform for executing scientific workflows. Its benefits and related provenance challenges as highlighted in the literature are also presented. Following this, a state-of-the-art survey to collect provenance in the Grid and the Cloud is presented. This helps in understanding the existing approaches and their shortcomings in line with the research objectives. Lastly, it presents the reproducibility tools and approaches discussed in literature. The literature is evaluated in the light of **Research Question 1** and **Research Question 2** of this research study.

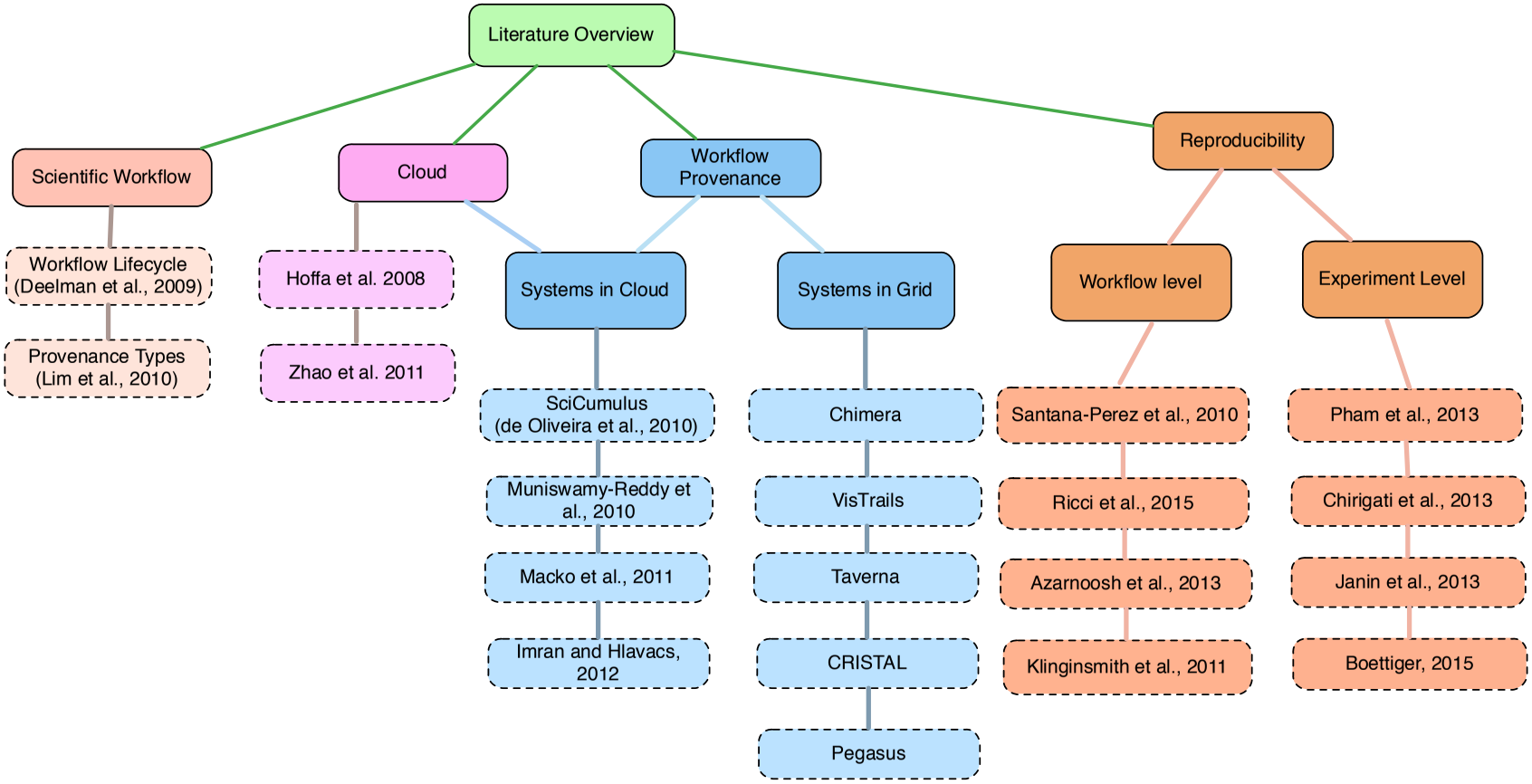


Figure 2.1: Literature survey roadmap

2.1 Scientific Workflows

This section introduces the scientific workflows and their applications in different modern scientific expeirments such as CMS or neuGRID. The life cycle of the workflow has also been presented in order to identify the key area of this research related to the workflow i.e. provenance. It then discusses different types of provenance information that is related to a workflow. This helps in further narrowing down the focus of this research study. The following paragraphs introduces the scientific workflow concept.

Scientific experiments such as CMS, LIGO or DNA analysis and projects such as neuGRID produce and process huge amounts of data. For instance, CMS produces an estimated raw data of size 5 Peta Bytes (PB) per year (Chatrchyan et al. 2008). Similarly, neuGRID's initial setup (Redolfi et al. 2009) estimated storing 20 Tera-Bytes (TB) of neuro-images. This data is then processed and analyzed using different tools and algorithms, such as neuroimaging algorithms in the case of neuGRID. These analyses require complex orchestration, which is performed using workflows. The workflows enable the researchers to collaboratively design, manage, and obtain results that involve hundreds of jobs or tasks accessing gigabytes of data. Here, a job is defined as an actual executable code/script used to access and process the given data. In this thesis, the terms job and task are used interchangeably.

The workflow concept was coined initially in late 1970s (Workflow-definition 2015). A workflow can be defined as a collection of jobs orchestrated in an order to achieve an overall goal. These tasks are arranged on the basis of their data or control dependency to perform complex data analyses such as the analysis of human brain images or the analysis of physics particles. Since these workflows involve a large number of jobs and require high computation, a distributed environment such as a Grid, and more recently the Cloud, is used for their execution.

The concept of workflows has been employed in many disciplines including scientific data analysis experiments, business process management and even within infrastructure environments i.e. the Cloud itself. The focus of this research is on the scientific workflows-based experiments. These workflows can also be divided into two further categories: (a) service oriented workflows, in which external services are invoked to perform intended operation; and (b) data oriented workflows, in which executable code/scripts performs the actual data analysis operations instead of invoking the external services. This research focuses on data-oriented scientific workflows for data analysis. In such workflows, the flow of data establishes the jobs' interdependency and is normally expressed as Directed Acyclic Graph (DAG).

2.1.1 Scientific Workflow and its Life cycle

The lifecycle of a workflow has been categorized (Deelman and Gil 2006; Deelman et al. 2009) into four main phases: composition (composing an abstract workflow), planning (mapping to available resources), execution (scheduling and individual job execution), and output retrieval along with provenance data. These four phases are discussed as follows:

1. **Composition:** Workflow composition allows a user to specify the phases and dependencies among tasks in an abstract way. Different authoring languages/tools, such as AWGL (Fahringer, Qin and Hainzer 2005) and Directed Acyclic Graph (DAG), where nodes represents individual jobs and edges represent the inter-job dependencies in XML (DAX), can be used to compose a workflow. The workflow authored at this stage has no computational resource information, it is added in the planning phases. While authoring a workflow, a user can also look into the provenance history to discover which job produced the desired output using a particular data input.

2. **Planning:** In this phase, a workflow is mapped to computational resources for execution. The mapping and planning procedures transform the given workflow to produce an executable workflow, which can be executed over a given infrastructure such as the Grid or the Cloud.

3. **Execution:** In this phase, the workflow jobs are scheduled to the execution resources based on the planned information in the workflow. These execution resources are provided through Grid infrastructure or more recently Cloud computing (Vöckler et al. 2011). In the planning phase, a user can specify the resources required to execute their jobs. If it is not specified then run-time resource selection is performed using appropriate scheduling approaches. During this phase, the status of a workflow and its jobs are continuously monitored to detect the job completion or failure. Once a job has finished successfully, its output is retrieved for the user.

4. **Provenance:** While retrieving the workflow's output, its associated provenance is also captured (Raicu et al. 2007). The provenance information includes job execution environment, parameters being used or input data provided to a job, job logs, intermediary outputs and the final output produced during the execution of a workflow. This information is becoming more and more important for scientists since it enables them to understand each phase of the workflow and then modify their workflows accordingly. Moreover, provenance information provides better insight about a problem by analysing the job logs and execution environment while debugging a job failure (Muniswamy-Reddy, Macko and Seltzer 2010). It also provides the history of data, which provides traceability and authenticity of the data being consumed or produced, which is essential in the Cloud. Moreover, reproducibility is also one of the main applications of provenance information (Davidson and Freire 2008). Therefore, this research study will focus on this phase of a workflow executed on the Cloud.

Cloud computing provides a paradigm-shifting, utility-oriented computing model in terms of the unprecedented size of data centre-level resource pools and on-demand resource provisioning mechanisms. This enables scientific workflow experiments capable of addressing petabyte-scale scientific problems (Zhao et al. 2011). As workflow execution is now being studied on the Cloud infrastructure, the aim of this research study is to capture the provenance of scientific workflows executing on the Cloud in such a manner that enables reproducibility of workflow execution on the Cloud.

2.1.2 Workflow Provenance Types

Provenance information collected for a workflow or its jobs can be divided into two broad categories (Lim et al. 2010; Freire et al. 2008). There is one further study (Freire et al. 2014) that provides a third type of provenance information; this third type is called Workflow Evolution. These categories are discussed below.

- **Prospective Provenance**: This maintains the description of an experiment which includes the computational job's specification (whether it's a script or a workflow) and corresponds to the steps (or recipe) that must be followed to generate a data product or class of data products.
- **Retrospective Provenance**: This captures the workflow steps executed as well as information about the environment use to derive a specific data product - in other words, this is a detailed log of a computational task's execution. Retrospective provenance does not depend on the presence of prospective provenance — for example, we can capture information such as which process ran, who ran it, and how long it took without having prior knowledge of the sequence of computational steps involved.
- **Workflow Evolution**: This captures the evolution history of a workflow structure itself. In doing so, it maintains all the changes applied to a workflow over its lifetime.

This research study in this thesis focuses on the Retrospective Provenance as it relates to the workflow execution and its execution environment. The aim is to investigate mechanisms to incorporate the Cloud layer (Infrastructure-as-a-Service (IaaS)) information in the existing provenance information of a workflow executing on the Cloud. This augmented provenance, which has been named Cloud-aware Provenance (CAP), will assist in achieving workflow reproducibility by re-provisioning similar resources on the Cloud infrastructure.

2.2 Cloud Computing

Cloud computing has emerged as a platform for large-scale data intensive computation. Its ability to provide an on-demand computing infrastructure with scalability enables distributed processing of complex workflows. This section provides information about the Cloud infrastructure, its recent use for scientific workflows and benefits. Since workflows have been executed on the Grid infrastructure, it is important to understand the difference between the Grid and Cloud environment and the challenges Cloud computing presents for workflow execution.

2.2.1 Cloud Architecture

Cloud computing is based on a layered architecture (Rimal, Choi and Lumb 2009). There are three main layers, as shown in Figure 2.2, including Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and IaaS layers.

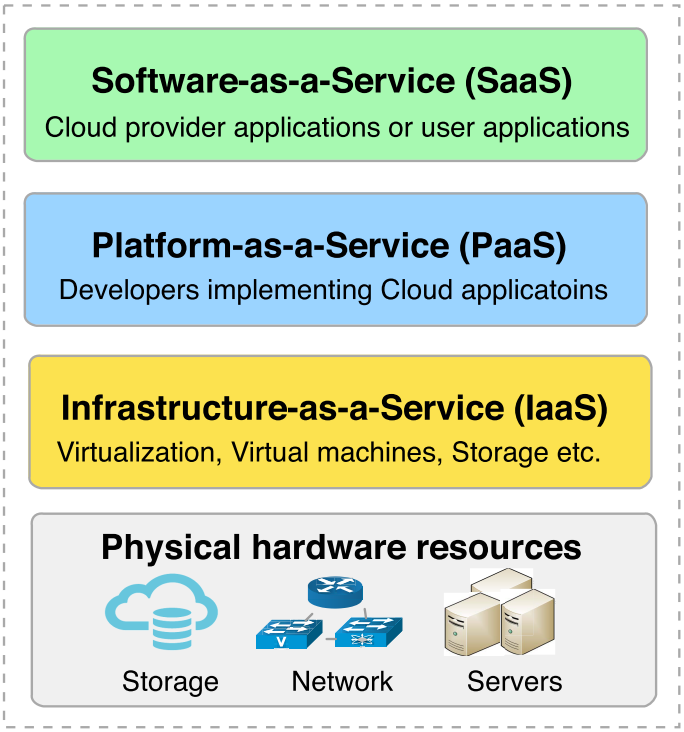


Figure 2.2: Cloud layered Architecture

The Infrastructure layer (i.e. IaaS) is also known as a virtualization layer on top of hardware resources. This layer provides a pool of storage and computing resources by dividing the physical resources using virtualization technologies such as Xen (Xen 2015) and KVM (KVM 2015). Amazon Elastic Compute Cloud (Amazon EC2), OpenStack[1] and Eucalyptus[2] are the leading examples of

[1]OpenStack: http://openstack.org
[2]Eucalyptus: https://www.eucalyptus.com/eucalyptus-cloud/iaas

IaaS. A computing resource in the Cloud is acquired by specifying resource configurations, which is composed of key parameters - RAM, vCPU, Hard disk and OS image. The platform layer (i.e. PaaS) is built on top of the infrastructure layer (i.e. IaaS) and it consists of an operating system, application middleware etc. It helps the developers to build their applications for the Cloud. It provides an interface for software developers to build new or extend existing applications, e.g., Google App Engine and Microsoft Azure (Imran and Hlavacs 2012). The software or application layer (i.e. SaaS) contains the set of applications provided by the Cloud provider or user using the Cloud infrastructure.

2.2.2 Cloud Adoption for Scientific Workflows

The possibility of using the Cloud as an execution infrastructure for scientific workflows has been explored (Hoffa et al. 2008) and it is fast becoming the future playing field for e-science research (Juve and Deelman 2010). An advantage of using the Cloud on scientific experiments is to provide scientists with access to a wide variety of resources without having to acquire and configure the computing infrastructure. There are a few initiatives, such as the Sloan Digital Sky Survey project and the Berkeley Water Centre, and high-energy physics community where the Cloud has been adopted (Williamson 2014; CERNVM 2015)

The execution of scientific workflows in the Cloud brings to the fore the need to collect provenance metadata, which is necessary to ensure the reproducibility of these experiments and the trust on the data being used in these experiments (Branson et al. 2012). Generally, an execution in Cloud based environments occurs transparently to the scientist, i.e. the Cloud infrastructure behaves like a black box. Therefore it is critical for scientists to know which parameters have been used and what data products were generated in each execution of a given workflow (Shamdasani, Branson and McClatchey 2012). With provenance information, scientists can debug experimental results and incorrect results can be checked in case something changes in their Cloud resources that might have affected the results (Muniswamy-Reddy, Macko and Seltzer 2010). Due to the dynamic nature of the Cloud, where resources are provisioned dynamically, the exact resource configuration should be known in order to reproduce the execution environment. Moreover, the resources on the Cloud are provisioned by specifying the resource configurations. For workflow reproducibility on the Cloud, such information should be known so that similar execution infrastructure can be re-provisioned on the Cloud. Due to these reasons, there is a need to capture and store provenance information for workflows executing in Cloud environments.

2.2.3 Benefits of Cloud Computing

Some of the key benefits, presented below, of using Cloud infrastructure for complex workflows execution has been expressed in many research papers (e.g. Zhao et al. 2014b; Strijkers et al. 2010).

- **On-demand resource provision**: The on-demand resource allocation mechanism in the Cloud can improve resource utilization and change the experience of end users for improved responsiveness. Cloud-based workflow applications can get resources allocated according to the number of nodes at each workflow stage, instead of reserving a fixed number of nodes upfront. Workflows executing on the Cloud can scale out and scale in dynamically, resulting in a fast turn-around time for end users. Similar findings have been presented by Strijkers et al. (2010). It shows that when applications require more resources at run-time, the Cloud is more flexible than the Grid infrastructure because requesting additional Grid resources by submitting jobs to the resource broker can take uncertain amount of time, which can negatively affect a workflow execution performance. However, with the Cloud, new resources can be provisioned instantly on demand.

- **Scale of Problem**: The scale of scientific problems that can be addressed by scientific workflows can be greatly increased compared to the Cluster/Grid environments, which was previously upbounded by the size of a dedicated resource pool with limited resource sharing extension in the form of virtual organizations.

- **Application deployment**: Application deployment can be made flexible and convenient. With bare-metal physical servers, it is not easy to change the application deployment and the underlying supporting platform. However with virtualization technology in a Cloud platform, different application environments can be either pre-loaded in virtual machine (VM) images, or deployed dynamically onto VM instances. A virtual machine (Goldberg 1974) is a running instance of a computer where resources such as memory and central processing units (CPUs) are allocated through virtualization software.

2.2.4 Difference between the Grid and the Cloud

It is important to highlight the differences here as they will serve as arguments for having a different mechanism for the Cloud environment. It would also serve as an argument-base for analysing the literature for Grid-based workflow management systems and the proposed provenance approaches. The Cloud differs from the Grid on several levels as highlighted by Foster et al. (2008); however this section only discusses the differences from a virtual resource point of view.

Most Grids use a batch-scheduled compute model, in which a Local Resource Manager (LRM), such as Condor (Tannenbaum et al. 2002) manages the compute resources for a Grid site, and users submit batch jobs (e.g. via GRAM) to request some resources for some time. Many Grids have policies in place that enforce these batch jobs to identify the user and credentials under which the job will run for accounting and security purposes, the number of processors needed, and the duration of the allocation. For example, a job could say, stage in the input data from a Universal Resource Locator (URL) to the local storage, run the application for 60 minutes on 100 processors, and stage

out the results to some FTP server. The job would wait in the LRM's wait queue until the 100 processors were available for 60 minutes, at which point the 100 processors would be allocated and dedicated to the application for the duration of the job. Once the job is finished, the user retrieves the output, however the compute resources will remain intact. On the other hand, in the Cloud where resources are provisioned on-demand, a user will send resource allocation requests, which specify the resource configurations, to the Cloud middleware and acquire the required resources. This makes the Cloud environment more dynamic and responsive. However, once the user job is finished, a user can release the resources and they will no longer exist. Therefore, it is important to capture the Cloud resource information in the collected workflow provenance in order to re-execute a workflow on the Cloud infrastructure.

2.2.5 Provenance Challenges in the Cloud

Provenance has not still been widely explored in Cloud environments, in which we need to deal with even more challenging issues, when compared with the Grid, such as tracking data production across different software and hardware abstraction layers within one provider (Foster et al. 2008). In addition, the scalability of Clouds requires much more scalable provenance systems that can handle the storage and querying of potentially millions of provenance records (Zhang et al. 2011). Considering only a single layer of provenance is not sufficient in many practical scenarios; one such example was at the core of the first provenance challenge[3], which required the integration of provenance to answer questions that necessitate an integrated view of provenance.

As the Cloud environment is different from other computing environments such as the Grid or Cluster and is often managed by a third party and is accessible over the Internet, therefore, provenance in the Cloud has also been discussed to achieve user trust on internal Cloud operations (Abbadi and Lyle 2011). This raises challenges in collecting provenance for the Cloud's internal processes. A similar argument has been presented by Kim-Kumar (Muniswamy-Reddy and Seltzer 2010) as he argued for including Provenance as first class data in the Cloud and that it should be stored alongside actual data on Cloud. However, this introduces the challenges of capturing such provenance and then storing and querying it. These arguments presented in (Abbadi and Lyle 2011; Muniswamy-Reddy and Seltzer 2010) are mainly in favour of tracking the services' activities of the Cloud itself, however this research study aims to track the provenance of a workflow executing on the Cloud infrastructure. Therefore, this research study does not focus on the internal activities of the Cloud services.

Since the scale of the workflow-based experiments has increased and can be supported through on-demand Cloud computing, this affects not only the data sizes that scientific applications need to handle, but also on the complexities of the applications themselves (Zhao et al. 2011). The scientific

[3] http://twiki.ipaw.info/bin/view/Challenge/FirstProvenanceChallenge

workflow applications on the Cloud can obtain resources allocated accordingly (dynamically) with the number of jobs at each stage of the workflow, instead of reserving a fixed number of resources. This necessitates a different mechanism to track provenance information on the Cloud that supports on-demand resource provisioning.

Generally, provenance systems in the Grid, workflow and distributed computing are either strongly part of the enactment engine or they use Application Programming Interfaces (APIs), which are enactment engine specific (Imran and Hlavacs 2012). Bose and Frew (2005) present a detailed survey of computational models and provenance systems in distributed environment, specifically workflow executions. However, there is a lack of support for provenance information in the Cloud. It has also been argued that in research environments, scientists are more interested in the overall process of execution and use step by step information to keep a log of sub-data and sub-processes to make their experiments verifiable, trust-able, reproducible and to have knowledge of the execution process (Imran and Hlavacs 2013).

Debugging, monitoring, and provenance tracking for workflows become increasingly difficult in a Cloud environment. Since compute resources are usually dynamically assigned and based on virtual machine. The environment that a task is executed on could be destroyed right after the task is finished, and assigned to a complete different user and task. Some workflow management systems also support task migration where tasks can be migrated to another resource (i.e. virtual machine) if there is problem with the resource on which the task was initially running (Zhao et al. 2014b).

Since provenance is also used to achieve reproducibility, it is essential to analyse existing workflow provenance approaches and their capabilities designed/proposed for the Grid as well as for the Cloud. This information has been analysed in the light of the aims of this research study and this aids in answering the **Research Question 1** - discussed in Section 1.4.

2.3 Key Elements in Provenance information

There has been an effort in designing a generic model for provenance data. Ram and Liu (2007) presents the seven Ws of data provenance. In the W7 model (Ram and Liu 2007), provenance is conceptualized as a combination of seven interconnected elements including “what”, “when”, “where”, “how”, “who”, “which” and “why”. Each of these components may be used to track events that affect data during its lifetime. *What* denotes an event that affected data during its lifetime; *When* refers to the time at which the event occurred; *Where* is the location of the event in terms of space; *How* is the action leading up to the event; *Who* are the agents involved in the event; *Which* are the devices used in job execution and they include programs or instruments (e.g. hardware) and application used in the event; and *Why* is the reason(s) for the events.

From the perspective of Cloud infrastructure, *Which* and *Where* become important as they can

provide information about the resources used to execute workflow jobs. Of these two, *Which* provides more meaningful information about a resource and its configuration. It is important to determine if this information is available in existing approaches for the Grid or the Cloud, and also essential to analyse if the provided information is sufficient to provision a new resource on the Cloud infrastructure.

2.4 Key Characteristics of Provenance Systems

Plenty of research has been done in supplementing provenance from various layers i.e. the workflow level and the system level, especially on the Grid. A few key characteristics suggested in literature (Glavic and Dittrich 2007; Cruz, Campos and Mattoso 2009), has been discussed in the following sections.

2.4.1 Capturing Mechanism

The two major approaches in capturing the provenance information are annotations and inversion (Cruz, Campos and Mattoso 2009). These are also known as the *eager* and *lazy* approaches. One approach focuses on computing provenance information when data is created, while the other computes provenance data when it is requested. These approaches are defined as follows.

- **Annotation:** The metadata comprising of the derivation history of a data product as well as process is collected as annotations and descriptions about sources data and processes. This is also known as the eager approach (Simmhan, Plale and Gannon 2005; Tan 2004).
- **Inversion:** The inversion method uses the property by which some derivations can be inverted to find the input data supplied to derive the output data. This is also termed as the lazy approach (Tan 2004).

The advantage of the inversion method is its compact representation of the provenance, compared to the annotation method. It is mostly used by databases. However, not all functions such as one-way hash algorithms can be reversed, thus the inversion approach is not universally applicable. Annotations give more flexibility in the richness of provenance metadata and the provenance need not be computed "just-in-time" like in the inversion method. A disadvantage is that the eager approach incurs in additional overhead to compute the output and to store annotations in the repository (Cruz, Campos and Mattoso 2009).

2.4.2 Granularity and Provenance Data Size

Provenance information can be collected at different levels. The more granular the provenance information is, the more details are available for debugging and analysis. For example, different granularity levels for a workflow in a Grid can be divided in the Workflow, Process/Task, OS and File levels. Each level captures provenance information to answer different user queries. In the context of the Cloud, granularity details can also include different Cloud layers such as Application level and Cloud infrastructure i.e. Virtualization level. The combination of these levels provides a comprehensive provenance view of the data consumed or produced on the Cloud.

Another important aspect of provenance information is its size as it can grow to be many times larger than the actual data (Chapman, Jagadish and Ramanan 2008). For instance, in a provenance use study (Groth et al. 2005), provenance information was collected for an experiment to determine the structure of protein sequences. The actual data used as input to the workflow was about 100Kb, however the provenance size was approximately 1MB, which is ten times the actual data size. There are a few systems (e.g. Barga and Digiampietri 2008; Scheidegger et al. 2007), which propose to reduce the provenance data size by avoiding repetitions and duplication of records or by eliminating the so-called orphan records from the system.

Granularity has a direct influence on the provenance storage size. More granular provenance data requires more storage space. In order to deal with the huge data size, one approach (Rozsnyai, Slominski and Doganata 2011) suggested the use of Cloud storage to store all the data, which is then processed to answer user queries.

2.5 Provenance Management in the Grid

Many workflow management systems have been proposed to execute scientific workflows on the Grid. Since provenance has been identified as a key element, these systems also offer the means to capture provenance information. The following sections provide a brief account of these systems and their approaches. Their collected provenance information has been critically analysed from a Cloud perspective in the light of the main aim of this research study.

2.5.1 Chimera

Chimera (Foster et al. 2002) is designed to manage the derivation and analysis of data for high-energy physics (*Griphyn* 2014) and astronomy (SDSS) (*SDSS* 2014) communities, which are data-intensive. Derivation means the transformation, which is the execution of a program, applied on a data object. It provides a Virtual Data Language (VDL) to insert and query provenance. It stores

provenance data along with metadata in a catalogue called the Virtual Data Catalogue (VDC), thus it uses a no-coupling storage approach (Glavic and Dittrich 2007). It collects process information along with the parameters used, the input data and produced data. Chimera stores this provenance information in a Chimera schema (a small part of which is shown in Figure 2.3), which is based on a relational database. Although the schema allows storing the physical location of a transformation i.e. the executable and its characteristics, it does not support the information of the hardware and software environment in which a transformation occurred.

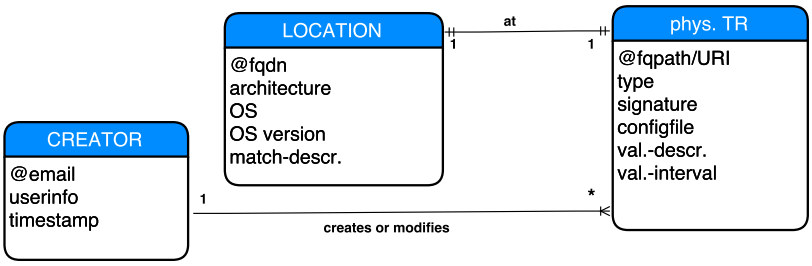


Figure 2.3: Chimera database schema dealing with physical location of transformation

2.5.2 VisTrails

VisTrails (Scheidegger et al. 2008) is a workflow and provenance management system that provides support for scientific data exploration and visualization. It not only captures the execution log of a workflow but also the changes a user makes to refine their workflow. It captures provenance information in three layers; (1) workflow evolution layer, (2) workflow layer, and (3) execution layer. The first layer captures the changes made in a workflow. The second layer comprises the workflow specification of an individual workflow and the third layer stores runtime information such as execution time and machine name related to a workflow. VisTrails represents prospective provenance as Python objects that can be serialized into XML and relations; it stores retrospective provenance in a relational database.

The provenance model of VisTrails allows storage of multiple versions of a workflow. In doing this, it stores only the changes applied to a workflow instead of storing complete versions of a workflow. This approach eliminates the redundancy and leads to better space utilisation. Provenance capturing in layers allows its schema to be normalised and it further reduces the redundancies. However, it does not support the virtualization layer (i.e. IaaS) of the Cloud. Although it can capture the machine name of where a job was executed, it still lacks information about the machine configuration, which is necessary to provision a similar resource on the Cloud.

2.5.3 Taverna

Taverna (Wolstencroft et al. 2013) is a workflow management system designed for the myGrid project (Stevens, Robinson and Goble 2003), as a pilot e-science project in the UK, which has been designed to provide middleware services to automate the execution of workflows in life science experiments (Zhao et al. 2004). It stores any prospective provenance information as Scufl specifications (an XML dialect) and retrospective provenance as Resource Description Framework (RDF) triples in a MySQL database. The retrospective provenance is collected in the form of a workflow log centred on the services in the workflow. The workflow execution log contains information regarding the service invocation, parameters being used, start and end times, and data products consumed and produced. This log is used to derive the data provenance of datasets generated during the workflow's execution.

Along with the metadata of data, it also maintains contextual and organisation metadata such as experiment information, project, and owner. This information helps in providing the context to the data provenance of a workflow. A data service, which is called myGrid Information Repository (mIR), has been designed over a relational database to store the metadata related to an experiment. Taverna does not provide a rich retrospective provenance related to a workflow execution. By analysing its database model, it has been found that it provides some information about the host when it uses SSH internally for remote machine access. However, this host information does not include resource configurations i.e. hardware or software.

2.5.4 CRISTAL

CRISTAL (Branson et al. 2012; McClatchey et al. 2014), developed at UWE and being used at CERN and in the neuGRID and N4U projects, is a provenance aware workflow instantiation system that is designed to orchestrate workflows, and capture and store their provenance information. It stores workflow specifications i.e. workflow descriptions in DAG and their instances, which are the workflows in execution, as its main entities. It uses an object model to express its main entities and an XML format to represent the workflow specifications. All entities within CRISTAL are described as XML and stored into eXist (Meier 2003), an XML database. Workflow orchestration is performed on a task-by-task basis and the provenance of each individual task is captured and recorded in its database. It not only supports the runtime provenance of a workflow, but it also maintains the history of workflow versions. However, it does not provide workflow planning and scheduling operations as these processes are performed outside its domain (discussed later in Section 3.5.2).

As of now the provenance information, retrospective provenance, it captures involves task inputs, outputs and the parameters that were given to the tasks along with the workflow versions. However, it does not provide the information about the execution environment and the configurations that were

used to execute these tasks. This information is particularly important in the context of the Cloud as the configuration and information about the execution environment is required to instantiate new virtual machines with the same specifications that would help in reproducing the results in terms of same performance and outputs.

2.5.5 Pegasus

Pegasus (Deelman et al. 2004) is a workflow management system designed for scientific workflows. It works with task oriented workflows, in which each job depends on an executable that is passed along with the job already configured and accessible on the infrastructure. This is the case with N4U where data and scripts files are available. Pegasus accepts abstract workflows, which are described in an XML based language called DAX, and converts them into executable workflow i.e. concrete workflow. In this conversion, abstract workflow specifications are mapped, via various transformations to a concrete executable workflow (Deelman et al. 2005). The mapping phase adds planning jobs (data management and clean up jobs) in the workflow along with its actual jobs. It then submits the jobs to the Grid or the underlying local submission system such as Condor cluster using DAG Manager (DAGMan) (DAGMan 2014).

For capturing provenance information, it uses a process-based capturing approach. A pegasus-kickstart activity[4] is used to launch the actual user job on the machine. This kickstart process monitors and logs provenance information, which is passed back to Pegasus for later processing. The captured workflow execution provenance includes job execution times, logs (stdout and stderr) and execution host (name or IP) information. Pegasus has also been used to evaluate the usage of Cloud computing for workflow executions (Vöckler et al. 2011; Juve et al. 2009). However, the captured host information is not sufficient to re-provision an execution resource on the Cloud. This will be discussed in detail later in this thesis.

A spin off project Pegasus/Wings (Kim et al. 2008) extends the DAX language for more rich semantic information about abstract workflows and for provenance handling, and is capable of recording the evolution of a workflow and storing all edit operations. Using this, Pegasus can also support prospective provenance. However, this research study is focused on the workflow execution provenance, rather than the provenance of a workflow itself (e.g. design changes) and how this information can be used to reproduce a workflow execution on the Cloud.

2.5.6 Kepler

Kepler (Ludäscher et al. 2006), a scientific workflow management system, has been designed to provide a workflow environment in which scientists can design and execute workflows. It aims to

[4]pegasus-kickstart: http://pegasus.isi.edu/mapper/docs/4.2/cli-pegasus-kickstart.php

keep track of all aspects of provenance in scientific workflows i.e. workflow evolution, data and process provenance. To enable provenance collection, it provides a Provenance Recorder (PR) component (Altintas, Barney and Jaeger-Frank 2006). In order to collect information that is generated during a workflow run, the PR implements several event listener interfaces. When something interesting happens, the event listeners registered with the specific 'concern' in question are notified, and take the appropriate action. For example, when the PR is notified that a data product is created, it can then associate the appropriate data lineage information with this data product and put it in the provenance store. Although, Kepler can track different aspects of provenance, in workflow execution provenance it only records the host_id, which could be an IP or hostname. It does not store other aspects of a machine, which are necessary in the context of the Cloud. In the Cloud, resource provisioning cannot be achieved only by providing the IP or hostname. A certain resource type or flavour (i.e. resource configurations including RAM, CPU, Hard disk) along with the required OS image are specified in a resource provisioning request to the Cloud middleware.

In 2012, Kepler performed an evaluation study (Wang and Altintas 2012) to incorporate the usage of Cloud resources for workflow execution. In doing so, a user needs to provide the type of virtual resource and the image he wants to provision on the Cloud. After provisioning the requested resource on the Cloud, Kepler then starts workflow execution. Though Kepler has this information, its provenance schema does not record it at the moment, which is the aim of this research work. Such information will enable reproducibility of workflow execution

One major disadvantage of using Kepler system is that Kepler users cannot develop workflows in a resource-independent way. Therefore, Kepler system does not provide benefits of workflow portability, optimization, and ease of design (Islam 2010). The Cloud resource information it maintains is taken manually from the users. Kepler does not provide a capturing approach that can collect such information from the Cloud.

2.5.7 Discussion

Although the aforementioned systems are designed to capture workflow provenance, they are not sufficiently generic to support the Cloud and the multiple layers in the Cloud. CRISTAL attempts to capture workflow provenance in detail, however it lacks the information related to the execution environment. Similar is the case with the provenance schema proposed by myGrid and Chimera. Moreover, these systems have been conceptually and practically designed for the Grid, and the Cloud and the Grid differ at various levels (Foster et al. 2008) such as execution infrastructure. Some of the systems such as Pegasus collects some information about the execution resource using the pegasus-kickstart command, which is executed alongside the job on the execution resource. Such an approach could influence the job execution performance. Kepler can provide some information about the Cloud resource but it depends on the user provided data, thus it does not provide any automatic mechanism to capture Cloud resource information. Some of these systems do provide information

Table 2.1: Summary of Grid-based Workflow Management Systems

System	Type (Prospective/Retrospective/)	Level(Workflow/Task/File)	Technique(Annotation/Inversive)	Granularity (Task, Data, Item)	Cloud Resource Infrastructure†
VisTrails	P+R	W	A	T	(p)
Pegasus	R	W	A	T	(p)
Taverna	P+R	W	A	T	(p)
Chemira	R	W	A	T	(p)
CRISTAL	P	T+F	A	Item	⋉
Kepler	P+R	W	A	Item	(p)
ReCAP	R	W	A	T	p

†⋉ = No support, (p) = Limited, p = Adequate support

Table 2.2: Summary of Resource information available in various systems

System	OS♯	CPU Count	RAM	Processor⁄	Hard Disk	Machine Identity†
VisTrails	ℓ	⊖	⊕	$\mathcal{D}$	⊖	δ
Pegasus	ℓ	⊖	⊕	$\mathcal{D}$	⊖	δ, ρ
Taverna	∘	∘	∘	∘	∘	ρ
Chimera	⊕	∘	∘	∘	∘	∘
CRISTAL	⊖	⊖	⊖	⊖	⊖	δ
Kepler	ð	⊖	⊖	⊖	⊖	δ, ρ
ReCAP	ð	⊕	⊕	$\mathcal{D}$, ⊛	⊕	δ, ρ

⁄$\mathcal{D}$ = Processor Description, ⊛ = CPU Speed
♯ℓ = OS Name, ð = OS Image
†δ = Machine name, ρ = IP
∘: Not Available, ⊕: Supported, ⊖: Not supported

about the operating system (OS) running on the execution resource but this information is mainly related to the Linux kernel running on that machine. Moreover, this information does not help in identifying an OS image on the Cloud which could be used to re-provision a resource on the Cloud. A summary of provenance characteristics of the reviewed systems is given in Table 2.1. This table also shows the intended characteristics (highlighted in green) of the provenance system, which will be designed in this research work.

With the help of these systems, one can capture workflow provenance, however more work is still needed to link this provenance information with the information of the Cloud IaaS layer to de-

velop a comprehensive provenance information which can be used for workflow reproducibility on the Cloud. The available resource information collected by the reviewed literature is given in Table 2.2. Cloud resources are dynamic and hierarchical in nature (Abbadi and Lyle 2011). Provenance capturing for a workflow can be even more difficult in the Cloud, since compute resources are usually dynamically provisioned, which could be destroyed right after the task is finished, and assigned to a different user and task (Zhao et al. 2011). Many of discussed systems do maintain either name or IP information of the execution resource that could be used while establishing the mapping between a job and a Cloud resource. The outcome of the intended research can enable the existing provenance capture such as that employed in Pegasus, which was designed for Grid, in capturing enriched workflow provenance from the Cloud.

2.6 Provenance Management in the Cloud

As mentioned earlier provenance in the Cloud has been a relatively recent idea (Muniswamy-Reddy, Macko and Seltzer 2009). There have been a few research studies (Muniswamy-Reddy and Seltzer 2010; Abbadi and Lyle 2011), which highlight the general challenges in building a provenance framework. The following are some of the approaches in literature for capturing and managing provenance information in the Cloud especially for the scientific workflows. Figure 2.4 illustrates the conducted literature survey on provenance management in the Cloud.

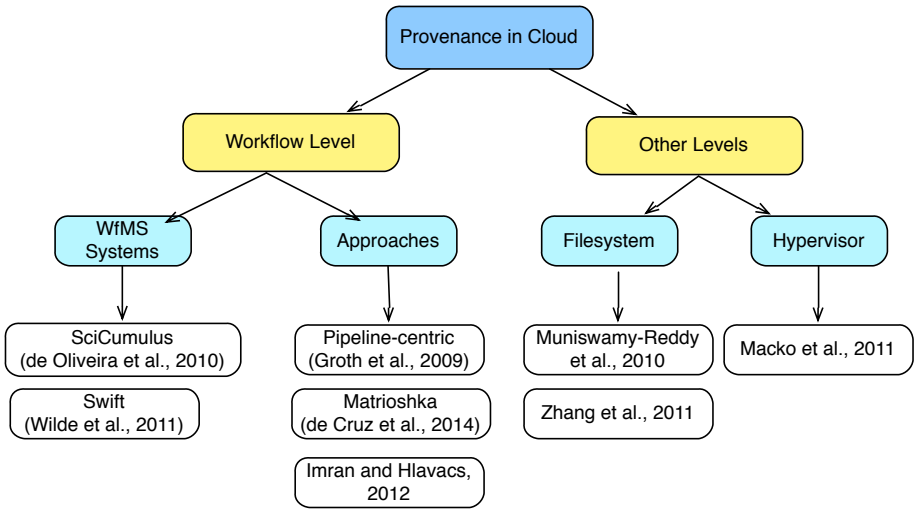


Figure 2.4: Literature survey of provenance management systems and approaches in Cloud

2.6.1 SciCumulus

SciCumulus (Oliveira et al. 2010) is a lightweight middleware designed to distribute, control and monitor the execution of scientific workflows in a Cloud environment, such as Amazon EC2. SciCumulus orchestrates the execution of workflow activities on a distributed set of virtualized machines. There are two main components, Dispatcher and Provenance Capturer, to perform these activities. The Dispatcher component launches the execution of workflow activities on the Cloud and communicates with the execution broker. The Provenance Capturer component is responsible for collecting provenance data in the Cloud. It collects provenance data, stored in virtual machines (VM), and transfers to a local repository. However, it does not collect the VM features that could be used to re-provision the execution environment. The collected host information includes the name and IP address, which is not sufficient to provision a resource on the Cloud. It can also store the resource type information if a user specifies this information in the configuration. It does not support an automatic Cloud resource information capturing approach.

2.6.2 Pipeline-centric Provenance

Groth et al. (2009) proposes a provenance model for workflow applications in which virtual machine images are a critical component. The idea behind this model is that, if a workflow is executed using a virtual machine (VM), then the VM image can be stored along with the provenance of the workflow. This enables the scientist to redeploy the VM and recreate exactly the same software environment needed for execution. The proposed provenance schema, represented in XML format, incorporates the references to the VM image location as VM size can vary and can be very large. It also stores the references to the input files, output files and workflow descriptions. This model assumes that the application software stack in a VM will remain the same; therefore, storing only the reference to a VM location would be sufficient for traceability. Storing information about the VM image solves the issue related to the software dependency. However, it does not solve the hardware characteristic of a resource because this model does not provide information about the type of virtual resource. On contrary, the aim of this research is to capture Cloud infrastructure details such that can be used to re-provision similar execution infrastructure for workflow reproducibility.

2.6.3 Swift

Swift (Wilde et al. 2011) is a parallel scripting system that allows for specifying, executing and analyzing scientific workflows with many computational tasks. Workflows are described in its own provided language, SwiftScript. By analyzing the inputs and outputs of the procedures defined in its script, the system determines data dependencies between them. This information is used to execute procedures that have no mutual data dependencies in parallel. The execution engine of Swift supports

environments commonly found in parallel and distributed systems. It is highly scalable and has been used for running large-scale scientific computations. Swift supports common execution managers for clustered systems and grid environments, such as Condor (Tannenbaum et al. 2002). It also supports Secure Shell (SSH) (Ylonen 1996), for executing jobs via secure remote logins to execution resources.

Swift can also be configured to execute jobs on the Cloud. In this configuration, it first deploys its worker processes on virtual machines over SSH. In a way, the Swift engine does not really know if the execution infrastructure is on the Cloud or not. Moreover, Swift does not handle the provisioning of resources on the Cloud. In one study (Maheshwari et al. 2013), Swift has been used on the Amazon EC2 Cloud. However, a separate Swift script was written to provision the required number of resources, already configured with Swift worker processes, on the Cloud and then the Swift engine started the experiment execution. Swift optionally produces provenance information in its log files, and this information can be exported to relational databases (Wilde et al. 2011; Gadelha et al. 2013). However, the collected provenance information does not provide any information about the Cloud resource used to execute the job.

2.6.4 Matriohska

The approach presented in (Cruz et al. 2014) has extended the Matriohska architecture (Cruz et al. 2008), which was initially designed for the Grid or clusters, to support Cloud information. It is based on three components; Provenance Broker, Provenance Eavesdrop and Provenance Repository. Provenance Broker receives the captured provenance metadata from the Cloud and stores it in a metadata repository. Provenance Broker is responsible for gathering provenance and associated metadata. The Provenance Eavesdrop component performs the task of collecting the metadata generated by the activities at the execution environment and sends it to the Provenance Broker, which stores it in the Provenance Repository, that is based on a relational database schema.

The provenance data schema encompasses Cloud specific features such as the concepts of virtualization i.e. the IaaS layer. In order to populate information about the Cloud resources, Matrioshka uses manifest files (in XML format) that provide information about the configurations. These files are provided by users unlike our approach that do not depend on user provided data. Moreover, the architecture of Matriohska requires installation of its components on virtual machines. The reported overhead caused by Matriohska is about nine minutes in executing an experiment. On the other hand, the approach proposed in this research aims to operate outside the virtual machine, thus it does not affect the job performance.

2.6.5 Provenance Framework for the Cloud Environment

Imran and Hlavacs (2012) propose an approach to collect provenance for the Cloud IaaS layer. The proposed approach targets Research Clouds, which rely on the open source technologies to provide an infrastructure (IaaS). The approach uses the Apache, Mule or similar frameworks and modifies Apache to intercept Cloud service events. This approach is tightly integrated with the Cloud middleware and thus can trigger appropriate handlers when an interesting event such as provision new resource occurs. Since it is part of the middleware, it does not have any information about the user's computational job let alone a workflow. Our aim is different from this proposed work as we want to work on workflow executing on the Cloud and to establish a mapping between workflow jobs and the Cloud resource. Moreover, this research work does not focus on the Cloud services and aims to work outside this realm. Furthermore, this approach targets Cloud provenance, however our work is focused on provenance of scientific workflows executing on the Cloud infrastructure layer (IaaS). It seems that this approach is internal to the Cloud services' processes.

2.6.6 File system level provenance in Cloud

A number of systems (e.g. Zhang et al. 2011; Tan et al. 2012) aim at providing data provenance at the file system level in the Cloud, mostly focusing on the system layer, which deals with data access in the Physical Machine (PM) and the VM. HP Labs propose a provenance management framework DataProv to capture provenance logs from the system layer. To achieve this, a distributed logging tool Flogger Ko, Jagadpramana and Lee 2011 has been used. Flogger is a file-centric logging tool developed to capture file operations at the VM and PM level. It helps in recording who, what (file operation), where (PM and VM) and at what time a file is accessed or transferred. Flogger requires kernel-level modules running in VM to capture file and network operations. It also requires a client module that can transfer the captured provenance to a server module running outside the VM.

There are a few other research initiatives (e.g. Muniswamy-Reddy, Macko and Seltzer 2009; Muniswamy-Reddy, Macko and Seltzer 2010) that aim to provide provenance at the file level using kernel modification. They achieve this through software hooks into the kernel and capture the OS calls such as file read, file write operations. By interrupting these file read and write operations, they manage to establish provenance of a file. The proposed protocol uses Amazon Storage S3 to store data along with its provenance metadata. There is another approach (Macko, Chiarini and Seltzer 2011) that tries to capture provenance at the hypervisor level, especially the Xen hypervisor. In this approach, they have extended the Xen hypervisor code to transparently collect OS level provenance from its guest VMs, without making any modifications in guest VM kernel unlike the approach presented by Muniswamy-Reddy, Macko and Seltzer (2010).

These aforementioned techniques are helpful in providing information about the operations on files being carried out in the Cloud. As these projects are designed to deal with file level provenance

information, they lack the provenance information of a workflow, which is the aim of this research study. Nonetheless, multi-layer provenance capturing requires provenance information from various layers: application, middleware and virtualization of the Cloud. In this regard, the Flogger tool can be helpful in capturing provenance information about a file at the infrastructure level. However, the research challenge here is to process this provenance information and then map it with the application level provenance of a workflow to provide a comprehensive view of the provenance to the user.

2.6.7 Discussion

In this section, different systems and approaches have been discussed that can execute and capture workflow provenance on the Cloud. The approach presented by Groth et al. (2009) only stores link to the OS image and does not include the characteristics of a virtual machine in its provenance schema. Moreover, it focuses only on the data provenance but neglects the information about the process that produces a certain output. There are some approaches such as Swift or SciCumulus that do not provide Cloud resource information in the collected workflow provenance. Then there is an approach (Imran and Hlavacs 2012) that deals with provenance of the Cloud services at the IaaS layer. It does so by intercepting the events for the Cloud services. Since this research study focuses on the workflow execution, this particular approach is not applicable. Furthermore, the approaches proposed by HP (Zhang et al. 2011; Tan et al. 2012) are file centric. They capture provenance of individual files hosted on a Cloud infrastructure. However, they lack the support for provenance information of a scientific workflow and interconnections of different files used in a workflow. The main motivation behind the provenance collection at a VM level or a hypervisor level, in these approaches, is to provide a way to look into the data leakages by monitoring file level activities in the Cloud. In summary, these approaches capture provenance information at different levels. However, they do not link them together to provide the Cloud-aware workflow provenance information for a user, which is the main objective of this research study. Furthermore, there are a few approaches such as Matriohska that do not provide Cloud information capturing approach rather they rely on user provided information. Another objective of this research study is to devise an approach that can automatically capture the information related to a Cloud resource and then link it with the workflow provenance. A tabular summary of the literature survey is provided in Table 2.3.

The critical analysis of the existing approaches shows that it is possible to collect provenance information at various levels i.e. workflow level, virtual machine, hypervisor as well as at physical machine level. In doing so, a few challenges have to be tackled as they have been highlighted in (Zhang et al. 2011). Provenance capturing at the VM level can affect the VM performance because continuous provenance collection might affect the overall performance, especially if applications are compute intensive. Similarly, virtual machines can be scheduled to any physical machine depending upon the scheduling algorithm used in a Cloud infrastructure. A virtual machine can be offline at any time and can then migrated automatically to another physical machine, in a manner totally trans-

Table 2.3: Summary of Cloud-based Provenance Approaches

System	Level(Workflow/Task/File)	Virtual Machine (Snapshot/Configuration)	Cloud Provenance Technique	Workflow Granularity	Cloud layer(Application/Infrastructure)
SciCumulus	W	-	User Provided	Workflow	A+I
Matriohska	W	C	User Provided	Workflow	A+I
Swift	W	-	User Provided + Log	Script	A
Pipeline-Centric	W	S	Manual	Workflow	A
HP	F	-	Log	N/A	-
ReCAP	W	C	Dynamic Mapping	Workflow	A+I

parent to a user. Provenance related to the physical machine could also become useful in providing a detailed data history. This requires a mechanism to link the provenance logs of virtual machines to the physical machine. However, this is complicated in real world scenarios i.e. scientists would not have direct access to the physical machines of a Cloud provider. They will be accessing the Cloud services through the online interfaces provided by the Cloud providers. Therefore, in this research study, the focus has been on provenance capturing of workflow at the application layer along with the virtualization i.e. infrastructure layer of the Cloud.

2.7 Reproducibility Tools and Approaches

Reproducibility has been defined as *"A result is reproducible if a member of the field can independently verify the result"* (Stodden 2011). The term reproducibility has also been treated as the ability for a third party who has access to the description of the original experiment and its results to reproduce those results using a possibly different setting, with the goal to confirm or dispute the original experimenter's claims (Missier et al. 2013). This definition is different from the first one, which is adopted in this research work. In this thesis, workflow reproducibility is treated as the ability of a researcher to re-execute the workflow with original data inputs and code, and over same execution resources on the Cloud, and produce the same results. Typically this means providing the original code and data, but does not imply access to proprietary software such as Matlab, or specialized equipment or computing power (C. 2010). In computational science, reproducibility often requires that researchers make code and data available to others so that the data can be analysed in a similar manner as in

the original publication. In the modern age, scientific applications have been growing in complexity and volume. Therefore, many scientists now formulate their computational problems as scientific workflows running on computing infrastructures provided by the Cluster, Grid and Cloud.

In earlier sections i.e. Sections 2.5 and 2.6, different systems and approaches designed for the Grid and Cloud environments to capture provenance information have been discussed. Their ability to capture the Cloud IaaS layer information, which is the focus of this research study, in the collected provenance has also been discussed. This section provides the literature survey on the tools and approaches using provenance information to reproduce an experiment execution. These projects and approaches to achieve reproducibility are then analysed with the focus on workflow reproducibility on the Cloud. Figure 2.5 illustrates the map of the reviewed literature.

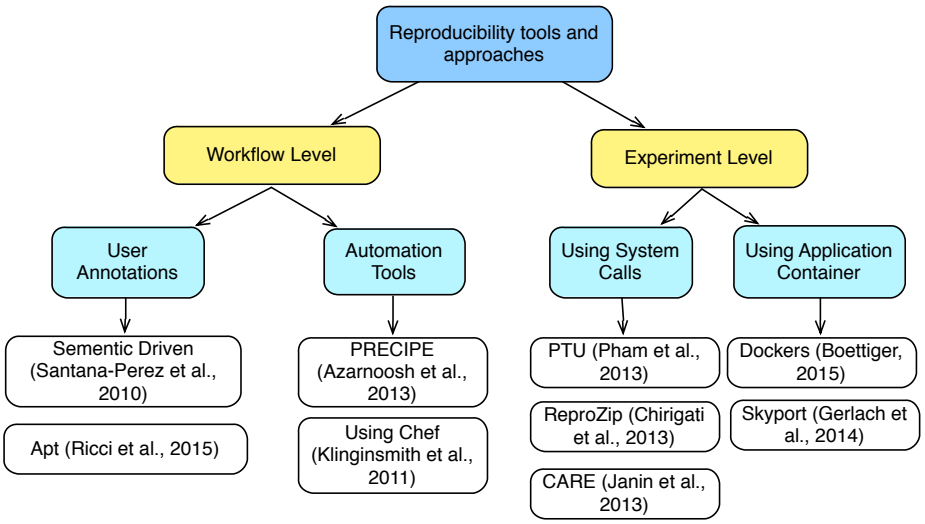


Figure 2.5: Literature survey of reproducibility tools and approaches especially in Cloud

2.7.1 Experiment Level Approaches

This section discusses the approaches and tools which have been designed to provide reproducibility and portability at the experiment level. These approaches mainly use system level calls or application container to achieve this objective. Following sections discuss these systems in detail.

2.7.1.1 Using System Calls

There are a few recent tools (e.g. Pham, Malik and Foster 2013, Chirigati, Shasha and Freire 2013, Janin, Vincent and Duraffort 2014) developed to provide experiment reproducibility by monitoring the system calls issued during a job execution. PTU (Pham, Malik and Foster 2013) is a tool that minimizes computation time during repeatability testing. The main objective of this tool is to

provide a mechanism that can assist in testing software programs that are submitted to conference and journals to conduct repeatability tests. It overcomes the issues of experiment setup and software environment (OS, Tools and libraries) for this purpose. Authors can use the ptu-audit command to build a package that includes their software program, input data and provenance trace of an initial reference execution. PTU relies upon ptrace, a system command, to monitor system calls such as read, write, socket, sys_io etc., for building a sequence of events' executions which become part of provenance. The provenance graph is then stored in a relational database v.i.z. SQLite. In order to create a package of original execution, PTU uses Code-Data-Environment (CDE) tool (Guo and Engler 2011), which is also Linux-based and relies upon ptrace. Users can select a subset of the package's processes for a partial deterministic replay based, for example, on their compute, memory and I/O utilization as measured during the reference execution. Using the provenance trace, PTU guarantees that events are processed in the same order using the same data from one execution to the next.

Another tool, ReproZip (Chirigati, Shasha and Freire 2013), deals with the computational environment and portability. ReproZip works by tracing the systems calls used by the experiment to identify which used files. It can capture or store the files/libraries used for executing an executable but it relies upon the available execution resource for that matter. The collected information along with all the used system files are zipped together for portability and reproducibility purposes. Similarly, CARE (Janin, Vincent and Duraffort 2014), a Linux-based tool, is designed to reproduce a job execution. It monitors the execution of the specified command to create an archive that contains all the material required to re-execute it in the same context. All these approaches are useful at individual job level but are not applicable to an entire workflow, which is the focus of this paper. They are more focused towards portability. Moreover, they do not maintain the hardware configuration of the underlying execution machine. Therefore, they are unable to determine the configurations of that resource which are important in the context of the Cloud. In the Cloud, a resource provisioning is achieved by specifying the resource configurations. Thus, a cloud infrastructure layer information is essential to collect. Moreover, packaging the physical infrastructure components limits the scope of applicability, as the packages require most of the target machine to be the same.

2.7.1.2 Using Application Container

Boettiger (2015) argued that computational work has become more and more integral to many aspects of scientific research, thus computational reproducibility has become an issue of increasing importance to computer systems researchers and domain scientists alike. Boettiger (2015) suggested an approach to use Dockers to provide cross-platform deployment, component reuse, sharing and archiving features. Using Dockers, they aim to address the challenges of recreating the development environment, dependencies issues and reuse. Similar approach was suggested by Gerlach et al. (2014). Since Dockers is tightly integrated with the Linux system and it shares the Linux

kernel with the host machine, and this makes it specific to a Linux kernel. Moreover, this approach attempts to address the deployment and portability issues by providing a self-contained application environment, however, this lacks support for Cloud infrastructure information. Moreover, this approach remains at the application level; it does not include the resource configurations, which is the focus point of this research work and important for acquiring a Cloud resource.

2.7.2 Workflow Level Approaches

In this section, a few approaches are discussed that aim to provide reproducibility at the workflow level. They achieve this by using use annotated information or automation tools with user provided infrastructure detail. The following sections discuss these approaches.

2.7.2.1 Apt: A Platform for Repeatable Research in Computer Science

Apt (Ricci et al. 2015) tackles the repeatable research environment problem and proposes that an appropriate environment in which to run an experiment is also required along with code and data. In doing so, it takes a hosted approach to building a repeatable environment: it consists of a cluster and a control system that instantiates encapsulated experimentation environments on that cluster. Apt uses profiles, which describe the experimental environments. A profile includes information about the code, data and resources, and this information is specified by a user. Apt's profiles capture an experimentation environment by describing both the software needed to run an experiment and the hardware (physical or virtual) on which the software needs to run. By providing a hardware platform for running these profiles, Apt essentially enables researchers to build their own testbed environments and share them with others, without having to buy, build, or maintain the underlying infrastructure.

The fundamental assumption behind this approach is that the surest way to build the foundation of a repeatable environment is to have a set of hardware resources that have known properties and are available, long-term, to all researchers. However, this assumption is not true in case of the Cloud environment. Moreover, the APT's profile approach is similar to Santana's semantic approach (Santana-Perez et al. 2014b) as both of them require user provided information about the experiment's resource configuration.

2.7.2.2 Semantic driven Approach

The approach presented in (Santana-Perez et al. 2014b; Santana-Perez et al. 2014a) discusses the importance of conserving an experimental environment to achieve reproducibility. This approach

relies on annotated semantic information provided by a user assigned to a workflow. The user-provided annotation is parsed and analysed to determine what type of resources are required and what type of software environment is required to be installed on them.

In this approach, a few semantics have been defined that enable it to acquire the required information. For instance, the *Workflow Infrastructure Description* is used to serves as the input of an algorithm, which is invoked whenever an infrastructure must be reproduced. This process also queries the state of the available Infrastructure Providers in order to obtain the resource availability. With this information about the former infrastructure and providers the algorithm generates an *Infrastructure Specification*, which defines a deployment plan detailing the resources to be created and how they must be configured. Finally, the Enactment System reads the Infrastructure Specification and carries out the actions defined on it over the Infrastructure Providers, producing the target infrastructure that the experiment would use to be executed.

This approach is similar to Apt's approach (discussed in Section 2.7.2.1) as both relied on user provided values to determine the needed execution environment. This work is similar to what is intended in this research study. Nonetheless, the proposed work in this research differs from the discussed approach because it attempts to combine workflow provenance from the workflow management system with the Cloud infrastructure information without taking any input from the user.

2.7.2.3 Automation/Deployment Tools: Chef

This paper (Klinginsmith, Mahoui and Wu 2011) argues that an experiment can be fully reproduced if both the infrastructure and software are configured in the same manner, and the data used within the original experiment is accessible. The presented reproducible experiment approach on Cloud is based on combining two layers i.e. the infrastructure layer and the software layer. The infrastructure layer is the layer that interacts with the API provided by a Cloud computing IaaS offering. The software layer deals with interactions on a running VM. It helps in installing and configuring software on the newly instantiated VMs. Using the infrastructure layer, it provisions the resources on the Cloud and then to create a similar software environment on the Cloud it uses a generic machine image within a Cloud. It then utilizes a configuration management tool, Chef (*Chef*), to build the fully configured VMs based on software installation scripts.

This approach can enable a user to recreate a similar environment (hardware and software) on the Cloud for an experiment's re-execution. However, this approach does not explicitly talk about the mechanism to retrieve a Cloud resource's hardware and software configurations. As it relies upon Chef, which uses recipes created by users, it can be assessed that this information comes from users. Moreover, it does not establish a link between a workflow job and the Cloud resource on which the job was executed, which is the aim of this work.

2.7.2.4 Automation/Deployment Tools: PRECIP

The Pegasus Repeatable Experiments in the Cloud (PRECIP) (Azarnoosh et al. 2013) library provides an API that enables researchers to provision resources on multiple clouds in order to perform their analyses. PRECIP can be used to set up experiments on academic clouds such as OpenStack Eucalyptus, Nimbus, and commercial Clouds such as Amazon EC2. It uses boto[5], a Python API, to interact with the different Cloud platforms. Though PRECIP claims to offer a flexible experiment management, it merely provides such a mechanism to collect workflow provenance. It provides an API that facilitates a user to provision resources on the underlining Cloud infrastructure and then to configure them using user-provided scripts. Moreover, it also provides a tag feature, which enables a developer to tag provisioned resource that helps in searching for required resources within the program. However, it does not provide any provenance information let alone provenance information of a workflow.

2.7.3 Discussion

In Section 2.7, recent tools and approaches to achieve reproducibility at experiment level or workflow level have been discussed. Many of them are designed with emphasis on computational reproducibility and portability. Tools such as PTU, ReproZip or CARE etc. work at the individual job level. These approaches will be useful in scenarios in which the complete experiment or analysis is composed of one job. Therefore, these approaches are not applicable to workflows which is the focus of this research study. Moreover, they also do not keep record of the hardware configurations of the resource used to execute a job. One possible use of such tools is to integrate them in executing a single workflow job and then use the produced outcome for reproducing that particular job execution. However, in a Cloud environment, relying only on this aspect is not sufficient because resources are yet to be provisioned by requesting the Cloud middleware and this requires information about the resource configurations, which are not considered in such tools.

The approaches such as Apt or Chef-based dealing with workflow level reproducibility rely upon user provided data. From this data, they deduce the required execution infrastructure and then they are able to re-provision or configure them. These approaches are close to the aim of this research study. However, they rely heavily on the user provided data instead of an automated approach that could discover the Cloud resource configurations used for workflow execution and then link it with the workflow provenance. The proposed work in this research differs from them it attempts to combine workflow provenance from the workflow management system with the Cloud infrastructure information without taking any input from the user. The approach discussed in (Azarnoosh et al. 2013) does not keep track of the workflow or the Cloud provenance information, which is the focus of this research study.

[5]Boto: https://boto.readthedocs.org/en/latest/

Table 2.4: Summary of Reproducible Approaches

System	Level(Workflow/Task)	Infrastructure	Software (Os/Libraries)	Purpose (Reproducibility/Portability)	Output (Zip/Directory/Provenance)	User Input(Required/No-Required)
PTU	T	-	L	R+P	Z	N
ReproZip	T	-	L	R+P	Z	N
CARE	T	-	L	R+P	Z	N
Containers Approach	T	-	L	R	NA	R
PRECIP	W	-	-	R	-	R
Apt	W	-	O+L	R	Z	R
Semantic	W	+	O+L	R	NA	R
ReCAP	W	+	O+L (using Image)	R	Provenance	N

2.8 Summary

This chapter discussed a scientific workflow and its lifecyle, and identified the Workflow Execution phase and its provenance as the focus of this research study. It then discussed Cloud architecture, its benefits and its increasing role in executing scientific workflows. It also highlighted important challenges related to provenance collection for workflows executing on the Cloud. Keeping the main aim of this research and identified challenges from literature in mind, this chapter then discussed the provenance management approaches (in Section 2.5) proposed for the Grid and also for the Cloud. Since many approaches such as Pegasus, Chimera and VisTrails were designed for the Grid, they consequently lack the support for the Cloud information in their collected provenance. In the absence of such information, re-provisioning of similar Cloud resources for workflow re-execution becomes challenging.

A few provenance approaches for the Cloud have been discussed in Section 2.6. The approaches proposed for the Cloud are very diverse since they provide information about file level access and Hypervisor level access which are useful from the point of view of providing data security on the Cloud. A few approaches have been proposed to discuss workflow execution and provenance collection on the Cloud. However, they either do not capture the Cloud's infrastructure information or they only provide such information with manual inputs from the user. Other approaches such as Matriohska rely upon provenance collection processes running inside the virtual machine along with the job, which caused an overhead of nine minutes in executing a workflow. Another approach provides only an URL of the OS image used in workflow execution and ignores other resource configurations such as RAM, CPUs etc., which are required in order to provision a resource on the Cloud. In

order to fill these gaps, this research study will aim to automatically collect the Cloud infrastructure information of the virtual machine on which workflow jobs were executed. This will eliminate the need of using the manual input from user describing the Cloud execution infrastructure. In order to avoid the performance overhead, this research will devise a provenance capturing mechanism that does not require provenance processes running outside the virtual machines.

To this end, this chapter discussed (in Section 2.7) the tools and approaches, which have been proposed to achieve reproducibility using provenance. The tools such as PTU, ReproZip or CARE rely upon system calls to monitor job execution traces and to produce a self-contained and self-sufficient reproducible package. Many of these systems are capable of achieving reproducibility of an individual job. However they lack the support for a workflow that is a collection of jobs, which is the focus of this research study. Moreover, the captured provenance does not include hardware details about that machine. A few approached proposed the use of application container such as Docker to mitigate the issue of software dependency and to provide execution environment portability. However, these approaches do not provide information about the virtual resource on which these containers will execute. The analysis of this literature survey provides a few insights as well.

From this literature review it has been established that the existing provenance capturing approaches do not provide Cloud information that could be used to re-provision the resources on the Cloud infrastructure for workflow re-execution. It was also found that some approaches rely only on capturing information about the OS image that provides software stack but they have ignored the resource configurations such as RAM, CPUs etc. of a virtual machine. A few approaches help in re-creating the execution infrastructure using the manual input provided by the user. These approaches do not provide any provenance capturing approach that could be used to collect Cloud infrastructure information at runtime. From this literature survey it has been found that a few approaches provide experiment reproducibility by using container applications such as Dockers or treating whole workflow as single executable job, however a scientific workflow represents a collection of tasks orchestrated in an order. With existing approaches, one can collect the workflow provenance and also the Cloud provenance at various levels. However, the interlink between a workflow job and the Cloud resource on which it was executed is missing. One of the main aims of this research study is to fill this gap and devise a mechanism to establish a mapping between a workflow job and the Cloud resource on which it was executed. This information can then help in re-provisioning similar execution resources on the Cloud and to reproduce a workflow execution on the Cloud.

In order to establish a mapping between a job and a Cloud resource, a machine's IP information can be considered as a potential mapping factor because many workflow management systems do maintain either host name or IP information (see Table 2.2). In the Cloud's IaaS layer, across one provider or for one user, no two machines can have same name or same IP at any given time. This means any running virtual machines should have unique IP or names. However, it is possible that a name or IP can be reused later for new virtual machines. The rest of the properties of a virtual machine accessible through the IaaS layer can be used by multiple machines at a time. This possible

mapping mechanism will enable to interlink workflow provenance with the Cloud infrastructure information. This

Based on the literature survey, **Research Question 1** has been answered. Limitations in the existing work have also been identified. It was also found that the link between workflow and Cloud information is missing and a possible mapping approach is outlined. This partially answers **Research Question 2**. The next chapter discusses the reproducibility requirements for a workflow executing on the Cloud in the light of existing work and also the Cloud environment. Chapter 3 will also provide the case study of N4U and its existing execution environment and its related provenance requirements.

Chapter 3

Workflow Reproducibility Requirements and the Proposed Architecture

In chapter 2, a literature survey was performed to analyse the existing systems and their capability to provide Cloud related provenance of workflows. Chapter 2 also analysed the approaches suggested in reproducing a workflow based experiments. However, they were not sufficient to achieve workflow reproducibility on the Cloud. This chapter aims to present reproducibility and to identify key requirements for workflow reproducibility on the Cloud. The structure of this chapter is follow. This chapter introduces computational reproducibility and the key points presented in the literature to achieve the reproducibility of experiments. Researchers have, for some time, been proposing guidelines to achieve experiment reproducibility as well as reproducible research papers. Based on the surveyed literature and Cloud context, this chapter then discusses a reproducibility model for workflow execution on the Cloud. This is followed by a brief introduction to N4U, a neuroscience project for imaging storage and analysis, and its requirements, particularly where related to workflow and provenance. The N4U case study helps in identifying provenance requirements for the neuroscience community. These requirements are then analysed in the Cloud context in order to understand the workflow reproducibility requirements on the Cloud using provenance information. It then discusses the existing workflow execution model used in N4U. Based on the provenance requirements in N4U and a clear absence of support for workflow execution on the Cloud in N4U, an architecture is proposed. The proposed system helps in capturing the Cloud resource information and maps it to the provenance of workflow executing on the Cloud infrastructure.

3.1 Computational Reproducibility

With the ever growing size of the experimental data and increasing complex processing workflows, the need for reproducibility has also become essential (Kanwal et al. 2015). These complex workflows are becoming more computational as they process and analyse the data to extract meaningful

information. Recently Sandve et al. (2013) listed a few simple rules for achieving reproducibility for computational experiments. There have been many other projects (e.g.Pham, Malik and Foster 2013; Ricci et al. 2015; Janin, Vincent and Duraffort 2014) (as also discussed in Chapter 2), which focused on achieving computational reproducibility. Multiple factors have been highlighted that hinder achieving computational reproducibility. Out of these, a reluctance to share the code used in generating results has been identified as a primary barrier to computational reproducibility in many domain sciences and this factor has nothing to do with the technological concerns rather social behaviours. Another factor that affects computational reproducibility is the ability to recreate or reconfigure the similar execution environment. This brings to the fore the environment (hardware and software) conservation as discussed by Santana-Pérez and Pérez-Hernández (2014). Hardware conservation means conserving the real object, due to its relevancy and the difficulty of getting a counterpart. This is mainly because the equipment, such as in a LHC experiment, is built once in a lifetime. The classical reproducibility approach in computational science aims to share the infrastructure by providing access to it for a community of users with the same interests (Santana-Perez and Pérez-Hernández 2015). This approach clearly fits into the physical conservation case; an organization or a set of them sets up an infrastructure (supercomputers, clusters, Grids) for a specific goal and allows some users to have access to their resources under some specific conditions. These are normally large and expensive infrastructures that require a lot of maintenance effort in the long term. These infrastructures have proved to be a significant contribution in computational science. However, Cloud computing offers a more dynamic environment and thus requires different approaches towards resource conservation. It becomes further challenging in the context of workflows, where a collection of tasks are executed on distributed resources provisioned on the Cloud infrastructure.

3.2 Levels of Reproducibility

Freire et al. (2008) have introduced three criteria to characterize the level of reproducibility of experimental results published in research papers. These points help in understanding the extent and ease to which an experiment can be reproduced. The three points are as follows.

- **Depth:** The depth indicates how much of an experiment is made available. The default today is to include a set of figures in a manuscript. Higher depths can be obtained by including: the script (or spread- sheet file) used to generate the figures in the research paper together with the appropriate data sets; the raw data and intermediate results derived during the experiments; the set of experiments (system configuration and initialization, scripts, workload, measurement protocol) used to produce the raw data; the software system as a white box (source, configuration files, build environment) or black box (executable) on which the experiments are performed.

- **Portability:** The level of portability indicates whether the experiments can be reproduced (a) on the original environment (basically the author of the experiment can replay it on his or her machine); (b) on a similar environment (e.g., same OS but different machines), or (c) on a different environment (i.e., on a different OS or machine).
- **Coverage:** This shows how much of the experiments can be reproduced: (a) partially, or (b) with full reproducibility. For example, for an experiment that requires special hardware to derive data, partial reproducibility can be obtained by providing the data produced by the hardware and the analysis processes used to derive the plots included in the paper.

Although the discussed points primarily focus on reproducible research papers, they still can be mapped onto workflow execution reproducibility. The *Depth* factor emphasises the amount of available information which is analogous to the granularity of the collected provenance. The *Portability* factor emphasises the capability to reproduce the same workflow execution on similar resources, which is the main focus of this research study. The *Coverage* factor determines the extent of reproducibility. Achieving full or strict reproducibility is hard to achieve as it involves various factors such as the availability of the software, libraries, executables, the load on the system, the hardware and version of the input data etc. The main idea is to be able to reproduce the entire execution of a given workflow whenever possible, not only the logical chaining of activities but also with the same data. To provide the strict reproducibility functionality, a system must guarantee that the original data are still accessible and that the corresponding activities are available (Lifschitz, Gomes and Rehen 2011). In this research study, partial reproducibility can be achieved by re-provisioning the similar resources on the Cloud infrastructure with an assumption that the data and processes of a workflow are available. However, strict reproducibility can also be achieved if the workflow, its jobs and dependent data are available and are not changed. Based on our understanding of reproducibility, next section points out a set of requirements identified for workflow reproducibility in the Cloud.

3.3 Workflow Reproducibility Requirements for Cloud

According to the current understanding of available literature, there is not a standard reproducibility model proposed thus far for scientific workflows, especially in a Cloud environment. However, there are some guidelines or policies, which have been highlighted in the literature to reproduce experiments. There has been one important effort by C. (2010) in this regard, but this mainly talks about reproducible papers and it does not consider the execution environment of workflows. The same concern has been shared by Santana-Perez et al. (2014b) that most of the approaches in the conservation of computational science, in particular for scientific workflow executions, have been focused on data, code, and the workflow description, but not on the underlying infrastructure, which is composed of a set of computational resources (e.g. execution nodes, storage devices, networking)

and software components. A recent study (Banati, Kacsuk and Kozlovszky 2015) concentrated on incorporating the infrastructure information in the collected provenance. In this section, a few basic points are gathered to present a set of workflow reproducibility requirements in the Cloud. These points also provide the basis for the proposed solution for workflow execution reproducibility on the Cloud. These points are discussed as follows.

1. **Code and Data Sharing**

   The need for data and code sharing in computational science has been widely discussed (C. 2010). Code must be available to be distributed, and data must be accessible in a readable format (Santana-Perez et al. 2014a). In computational science, particularly for scientific workflow executions, it is emphasized that the data, code, and the workflow description should be available in order to reproduce an experiment. In the absence of such information or data, experiment reproducibility cannot be achieved because different results would be produced if the input data changes. It is also possible that the experiment cannot be successfully executed in the absence of the required code and its dependencies.

2. **Execution Infrastructure**

   A workflow is executed on an infrastructure provided by the Grid or the Cloud. The execution infrastructure is composed of a set of computational resources (e.g. execution nodes, storage devices, networking). The physical approach, where actual computational hardware are made available for long time periods to scientists, often conserves the computational environment including supercomputers, clusters, or Grids (Santana-Perez et al. 2014b). As a result, scientists are able to reproduce their experiments in the same hardware environment. However, this luxury is not available in the Cloud in which resources are virtual and dynamic. Therefore, it is important to collect the Cloud resource information in such a manner that will assist in re-provisioning of similar resources on the Cloud for workflow re-execution. This will enable a researcher to recreate a virtual machine with similar resource configurations. Banati, Kacsuk and Kozlovszky (2015) also emphasized on the need to incorporate infrastructure information as part of the workflow provenance.

   From a resource provisioning as well as a performance point of view, the following factors are important in selecting appropriate resources especially on the Cloud. These factors include: RAM, vCPU, Hard Disk, CPU Speed in MIPS. All these factors contribute to the job's execution performance as well as to its failure rate. For instance, consider a job that requires 2 GB of RAM during its processing. This job will fail if it is scheduled to a resource with less available RAM. Moreover, it could also affect its performance if more and more data is processed from hard disk. Similarly, vCPU (virtual CPUs, meaning CPU cores) along with the Million Instructions per Second (MIPS) value directly affect the job execution performance. In a study (Vöckler et al. 2011), it was found that the workflow task durations differ for each major Cloud, despite the identical setup. It was suggested that lower/different CPU speed,

and a poor WAN performance could be one factor for different or slow workflow execution times.

Hard disk capacity also becomes an important factor in provisioning a new resource on the Cloud. It was argued that building images for scientific applications requires adequate storage within a virtual machine. In addition to the OS and the application software, this storage is used to hold job inputs and output that are consumed and produced by a workflow job executing on the VM (Vöckler et al. 2011).

Out of these factors, current Cloud offerings only support the provision of resources based on RAM, vCPU and Hard Disk. These factors are combined and named as instance type (e.g. in Amazon EC2), or flavour (e.g. in OpenStack). The MIPS information is not provided as a parameter for acquiring a resource. Therefore, the proposed architecture would take these three factors along with the software environment (discussed below) into consideration. Nonetheless, the efficacy of having MIPS information in the collected provenance will be shown through our results (discussed in Chapter 6). This will aid in providing a motivation and envisioning a future possibility in which the Cloud Providers will start this as a configurable parameter of a resource.

3. **Software Environment**

   Apart from knowing the hardware infrastructure, it is also essential to provide information about the software environment. A software environment determines the operating system and the libraries used to execute a job. Without the access to required library information, a job execution will fail. For example, a job, relying on a MATLAB library, will fail in the case where the required library is missing. One possible approach (Howe 2012) to conserve software environments is thought to conserve the VM that is used to execute a job and then reuse the same VM while re-executing the same job. One possible mechanism is to create snapshot of virtual machines for each job, however the high storage demand of VM images poses a challenging problem (Zhao et al. 2014a). In the prototype proposed in this research study, the VM is assumed to present all the software dependencies required for a job execution in a workflow. Therefore, the proposed solution will also retrieve the image information in building a virtual machine on which the workflow job was executed.

4. **Workflow Versioning**

   Scientific workflows are commonly subject to a reduced ability to be executed or repeated, largely due to the volatility of the external resources that are required for their executions (Gómez-Pérez et al. 2013). Capturing only a provenance trace is not sufficient to allow the computation to be repeated – a situation known as workflow decay (Roure et al. 2011). The reason is that the provenance systems can store information on how the data was generated, however they do not store copies of the key actors in the computation i.e. workflow, services, data. Workflow versioning along with other provenance information has been suggested to

achieve reproducibility (Woodman et al. 2011). Recently, Sandve et al. (2013) have suggested archiving the exact versions of all programs and enabling version control on all scripts used in an experiment. This is not supported in the presented prototype because the focus of this research study is on the execution aspect of a workflow. Nonetheless, it can be incorporated in future by using CRISTAL since it can track the evolution of its stored items (Branson et al. 2012). Moreover, the versioning also suggests tracking the evolution of a workflow design itself. This means to track the changes made in the workflow composition by adding or removing or modifying a workflow component in its design. As researchers use a workflow management system to carry out their computations the workflows evolve as the research evolves and this workflow evolution needs to be tracked to identify the results created by a specific workflow instance across time (Barga et al. 2010). Scientists can trace their research and associated results through time or go back in time to a previous stage and fork a new branch of exploration. Since the focus of this research work is on the workflow execution phase, this aspect has consequently not been discussed in detail, however, the original workflow description along with its associated files has been stored to support workflow reproducibility of the same workflow.

5. **Provenance Comparison**

   The provenance of workflows should be compared to determine workflow reproducibility. The comparison should be made at different levels; workflow structure, execution infrastructure, and workflow input and output. A brief description of this comparison is given below:

   (a) Workflow structure should be compared to determine that both workflows are similar. Because it is possible that two workflows may have a similar number of jobs but with a different job execution order.

   (b) Execution infrastructure (i.e. the software environment and resource configuration) used for a workflow execution should also be compared.

   (c) Comparison of the inputs and outputs should be made to confirm workflow reproducibility. There could be a scenario in which a user repeated a workflow but with different inputs, thus producing different outputs. It is also possible that changes in job or software library results into a different workflow output.

   In general, the provenance approach used by most Workflow Management Systems (WMS) such as Kepler or VisTrail enforces this strict reproducibility requirement for relatively small amounts of consumed and produced data, or just for primitive data. The most common strategy is to save all consumed and produced data into a relational database, together with a link among data and the corresponding execution (Lifschitz, Gomes and Rehen 2011). However, this approach is not feasible for large data files.

There are a few approaches (e.g. Missier et al. 2013) that perform a comparison on workflow provenance graphs to determine differences in reproduced workflows. The proposed approach

in this research study incorporates the workflow structure and infrastructure along with output comparison to determine the reproducibility of a workflow. The important difference from Missier's approach is the comparison of Cloud infrastructure information in the provenance graph.

6. **Pricing model**

   This point can be important for experiments in which cost is also a main factor. However, it can also be argued that this information is not trivial for an experiment due to strong industry competition between big cloud providers such as Amazon, Google and Microsoft etc., which can bring prices down. Having said this, one still cannot deny the fact that a cost is associated with each acquired resource on the Cloud, thus making this factor important to be focused on. The pricing factor has been used in various studies to conduct the feasibility of a Cloud environment for workflow execution (Deelman et al. 2008a). In this study, the cost factors for various resources such as compute and storage have been evaluated for workflow execution. The pricing information has also been used in cost-effective scheduling and provisioning algorithms (Abrishami, Naghibzadeh and Epema 2013; Malawski et al. 2015).

   Abrishami, Naghibzadeh and Epema (2013) use billing information as a key parameter for devising a workflow scheduling algorithm on the Cloud to execute a workflow within a given deadline. This information can also be used in provisioning resources cost-effectively on the Cloud (Malawski et al. 2015). Therefore, this pricing information, if collected as part of provenance, can help in reproducing an experiment within the similar cost as was incurred in earlier execution. However, one must keep this in mind that the prices are dynamic and subject to change and it depends entirely on the Cloud providers. For a static environment, in which cost does not change rapidly, such information can be helpful. Therefore, this point is captured as part of the Cloud-aware provenance and incorporated in the proposed design of ReCAP (discussed later in Section 3.8).

3.4 Provenance Correctness and Completeness

As discussed in Chapter 1, this research study also aims to identify a means to specify and measure the correctness and completeness of the collected provenance (see **Research Question 3**). Provenance can often be incomplete with resulting gaps and errors in the provenance record. This may be a result of the unreliable protocols between application and provenance storage or the act of stitching together provenance traces through time (Missier et al. 2012). It is also possible that a workflow could not complete its execution due to failed jobs, thus the resultant provenance of the workflow is incomplete. In literature (Dai et al. 2008; Simmhan and Plale 2011), provenance data has been used to assess the quality of data, however it is also important to measure the quality of the collected provenance itself to ensure that captured provenance traces can be used as intended. One such work in this regard is presented by Cheah and Plale (2012).

According to Cheah and Plale (2012), the quality of provenance data relies upon two key dimensions namely Correctness and Completeness. These two dimensions are defined as:

1. **Correctness:** The provenance correctness denotes the extent to which provenance data are correct and free of error. This dimension encompasses attributes such as the accuracy, unambiguity, consistency and homogeneity of the provenance. Correctness is assessed primarily through contextual analysis of the provenance graph.
2. **Completeness:** The provenance completeness denotes the extent to which provenance information is missing. Completeness is assessed through structural analysis of the provenance graph.

The model proposed in (Cheah and Plale 2012) suggests performing the structural analysis on the provenance graph in order to measure the completeness. The structural analysis is performed by comparing the nodes and edges of a graph to a reference graph, which is the workflow description. This approach assumes that the generated provenance graphs are Directed Acyclic Graphs (DAGs) and adhere to an execution template i.e. a workflow description. This approach uses annotation analysis and timestamp analysis to perform the contextual analysis in order to measure the correctness.

The above approach aims to identify and fill the missing provenance information with possible values. There is another approach (Zhao, Gomadam and Prasanna 2011) that aims to predict the missing provenance information in reservoir engineering using semantic data. This approach uses annotated data items and establishes a connection sequence between them. By analysing the annotated historical datasets with complete provenance information, it captures and evaluates the semantic associations that may imply identical provenance or predict the missing information.

Since this research study is focusing on workflow execution provenance and its applicability on workflow reproducibility on the Cloud, the correctness is redefined as *"the extent to which the Cloud specific information is accurately acquired in the Cloud-aware provenance"*. This means that this research study aims to capture and store the Cloud resource information collected from the Cloud infrastructure using Cloud APIs without tempering or modifications. Since this work will not rely upon the user provided annotations, thus the contextual analysis using the annotated data or semantic data is outside the scope of this research work. Nevertheless, the correctness analysis should incorporate the presence of identified Cloud specific parameters in determining the correctness of the collected provenance. To achieve this, it is necessary to capture the Cloud resource configuration, which is composed of vCPU, RAM, Hard Disk and OS image. If all these parameters are collected for a job and a mapping between the job to the Cloud resource is established, this means the collected Cloud-aware provenance is correct and unambiguous, and can be relied upon. Another aspect to look into the correctness is to verify the values in the collected provenance of Cloud resource. In this thesis, this has been achieved by relying on the Cloud middleware to provide the correct resource

information. Since this information is taken directly from the Cloud middleware and has not been tempered with, it is thus assumed that the collected Cloud resource information is correct. This information is then parsed to extract the required aforementioned Cloud resource parameters such as vCPU, RAM and OS image etc.

In order to perform a completeness analysis of the provenance, the research work in this thesis has devised a mechanism to compare a given provenance graph based on the structural analysis i.e. nodes and edges of the provenance graph, as suggested in (Cheah and Plale 2012). In this analysis, the execution infrastructure information is also incorporated in the provenance completeness analysis. By combining the workflow structure with the collected Cloud provenance, it will determine the completeness of the captured Cloud-aware provenance.

3.5 Case Study: N4U

The use of workflows to perform complex computation in processing large datasets has been adopted in many scientific domains such as astronomy and neuroscience. In neuroscience, one such project is N4U[1]. It is the extension project to neuGRID (Munir et al. 2013) that aimed to help neuroscientists carry out high-throughput imaging research for individual patient diagnosis especially for Alzheimer's disease. Using its infrastructure in 2009, neuGRID was able to extract an Alzheimer's disease biomarker (3D cortical thickness with Freesurfer) from 6,500 MR scans within two weeks whereas the process can take five years using previous computational imaging analysis techniques. To achieve high-throughput imaging analysis of complex design, N4U uses the concept of pipelines i.e. workflows. One such example is the ReconAll workflow.

The pipeline defines the set of activities that a researcher wants to perform over a given set of selected dataset files. This pipeline is executed on a Grid infrastructure, which is maintained by the N4U consortium, through CRISTAL. CRISTAL assists users in defining their analysis by allowing them to choose a pipeline and required input files. This user analysis is then submitted through a Pipeline Service for execution. The Pipeline Service accepts a workflow defining all the jobs, with their arguments, inputs and output files and submits it to the N4U Grid. A user can monitor the status of his workflow and can retrieve the output once the workflow is finished. Along with the output, a user can also retrieve provenance information such as start time, finish time, job logs, and exit code etc.

3.5.1 Workflow Provenance Requirements in N4U

Reproducibility has been recognized as an important concern in neuroscience/neuroimaging research (Garijo et al. 2014; Gronenschild et al. 2012). Reproducibility in neuroimaging studies may be dif-

[1] https://neugrid4you.eu/

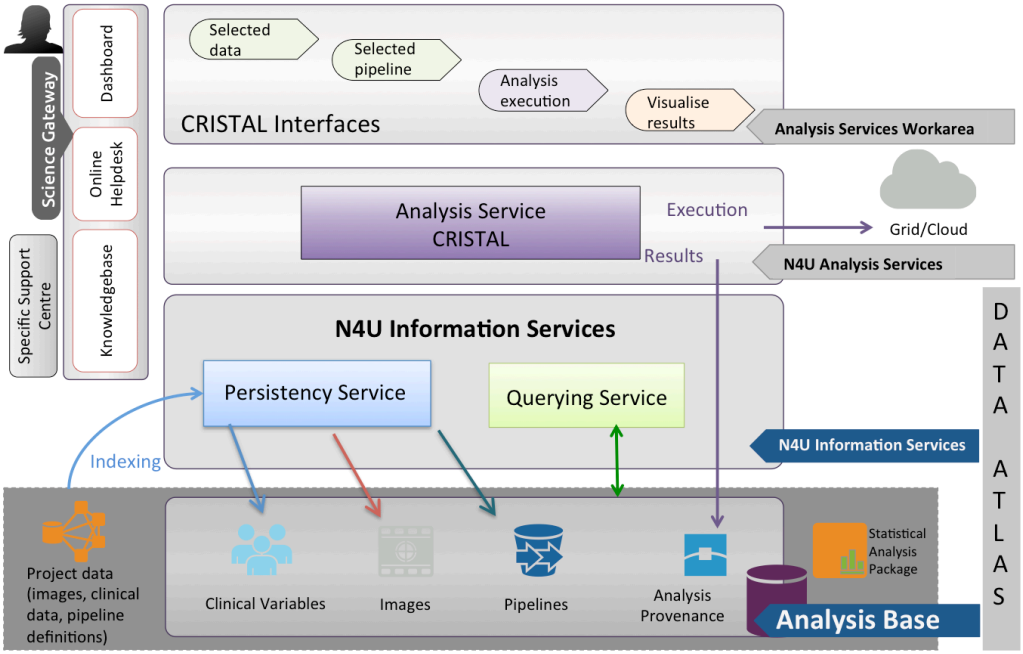


Figure 3.1: A virtual laboratory for conducting analysis in N4U

ficult to achieve between laboratories as journal space constraints may limit the ability of researchers to report the occurrence and ordering of complex analysis steps with sufficient detail to allow a new user to re-execute the analysis in exactly the same way. Deficits in the ability to reproduce analyses using new data add variability to results among labs, making interpretability of results more difficult.

In the neuGRID project, the following user requirements related to workflow execution and result validity were captured during the requirement gathering phase (Spulber, Damangir and Wahlund 2012).

1. **Capture Workflow Execution Provenance**

   In the user requirements, it has been found that the users are keen to capture provenance data related to workflow execution. This information includes storing workflow setup and input parameter and information on an intermediary execution step. It also requires storing output and error messages produced as a result of a workflow execution.

2. **Store and Track History of Workflow Execution**

   This requirement intends to store information about an executed workflow, its number of executed jobs, and also to track the number of successful of unsuccessful workflows.

3. **Validate a Workflow using Provenance**

   It is part of the user requirements to devise a mechanism that can assist in validating a workflow execution using provenance data. It also requires a means by which a user can access the

provenance data for further investigation. Since workflows can be shared among the workflow authors, the associated provenance data can assist in understanding the existing work and also in reproducing the workflow execution.

The workflows in N4U are executed through CRISTAL that keeps history of the executed workflows and stores the output and error logs produced during workflow execution. It also collects provenance about the workflow through the Pipeline Service. As of now, the Pipeline Service only provides information about the job outputs, and job start and finish times. In relation to execution host information, it only provides a host name, which is insufficient in order to re-provision resources in the Cloud environment. It has to be noted that N4U provides a Grid infrastructure and all its related systems are also Grid-based. The workflow execution in N4U Grid infrastructure and the collected provenance information is discussed in following section.

3.5.2 Workflow Execution in N4U

The N4U Virtual Laboratory (shown in Figure 3.1) provides its users with an Analysis Service Workarea (Munir et al. 2014), accessed by a graphical Dashboard, where they can specify scientific analyses to be carried out. A user defines an analysis by selecting images from the datasets, choosing a specific algorithm/pipeline and submitting a job to carry out the analysis. In N4U, a workflow along with its inputs (argument and datasets) submitted for execution is called an analysis. This analysis request is submitted to the CRISTAL, a description-driven system (McClatchey et al. 2013). CRISTAL creates an item for this analysis in its database, an XML database using eXist db (Meier 2003), to provide provenance support. To execute a user provided analysis, a component GridBroker is written in CRISTAL that continuously looks for new jobs in CRISTAL. The CRISTAL software orchestrates the execution of the analysis, tracks its progress and gathers provenance data for each step (or activity) of the pipeline.

The GridBroker is responsible for constructing the workflow specification for each user analysis, for submitting a workflow on a user's behalf, for monitoring the workflow status, and for retrieving the workflow output (job stdout and stderr) once it has finished. It retrieves a new user analysis request from CRISTAL and constructs a workflow specification compatible with the Pipeline Service used in N4U. The Pipeline Service is responsible for parsing the submitted workflow/job specification for consistency checks, and for identifying suitable resources on the N4U Grid infrastructure for its execution. The Pipeline Service creates a scheduling plan for the workflow and schedules the workflow to the available resources on the N4U infrastructure. Figure 3.2 illustrates the analysis submission in the N4U environment.

After the workflow has completed i.e. the job has executed, whether successfully or unsuccessfully, the results of the execution (output, error information, new data, etc.) and related provenance

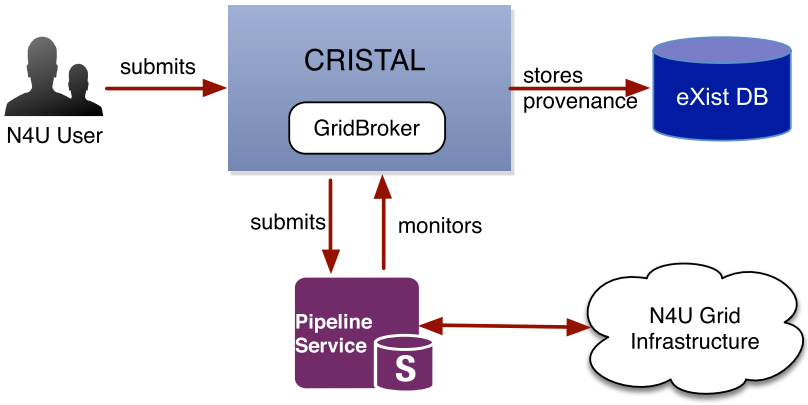


Figure 3.2: A N4U User submitting analysis to N4U infrastructure through CRISTAL

information are passed back to CRISTAL and stored in the database. In N4U, the provenance of a user analysis provides information about the pipelines and datasets used in an analysis along with its execution information such as job logs and outputs, exitCode, reason, execution state and execution times etc. The collected provenance information, in the XML representation shown below, has information about the execution time (*When* of W7 model (Ram and Liu 2007)) and also execution host (*Which* of W7 model (Ram and Liu 2007)). However, it does not provide further information about the configuration of that resource used to execute the job, which is the focus of this research study. It is also to be noted that one of the N4U requirements (discussed in Section 3.5.1) is related to workflow validation and reproducibility using provenance data. With the provenance information captured in N4U, it is not possible to reproduce a workflow execution in the Cloud environment. Moreover, the Pipeline Service is integrated with the Grid since N4U infrastructure is Grid-based and thus it does not provide support for a Cloud environment. Furthermore, no such APIs are provided in N4U to retrieve such information about its infrastructure.

In N4U, the Cloud has been used internally by administrators to deploy the Grid infrastructure on virtual machines. The access to the Cloud middleware is not provided to the users or the developers. The Cloud is used only to expose the physical infrastructure as virtualised resources. As soon as the new resources are required as per the need of the project, a new virtual machine is provisioned and the Grid middleware is installed on it. The compute resources in N4U are exposed in a static manner (a Static use case; discussed later in Section 4.1) as is the case in the Grid infrastructure. However, the focus of this research is on the Cloud environment, which is dynamic and resources are provisioned on demand.

Since this research study is focusing on the Cloud as an execution infrastructure, the use of CRISTAL for executing a workflow on the Cloud environment has been evaluated. However, currently CRISTAL does not provide the mapping, scheduling and data management operations during a workflow execution. For this, it relies upon the provided Pipeline Service, which is developed for the Grid infrastructure of N4U. This is why another workflow management system, Pegasus, has been identified to be used in the proposed prototype for demonstrating the proof-of-concept work.

Listing 3.1 Workflow job output represented in XML by the GridBroker in N4U

```
<GridJobStatus>
  <workflowID>
[GridWorkflowHandler]-{[wms://wms.maatg.eu:7443/glite_wms_wmproxy_server]-
[https://lb.maatg.fr:9000/Pybe2vLr7KLjlhpnUG9NyQ]}
  </workflowID>
  <state>DONE</state>
  <exitCode>0</exitCode>
  <reason>null</reason>
  <startDate>2013-06-25T00:40:44.000+01:00</startDate>
  <createdDate>2013-06-25T00:37:27.000+01:00</createdDate>
  <endDate>2013-06-25T00:49:19.000+01:00</endDate>
  <stdout><![CDATA[before:/opt/exp_soft/neugrid (standard stdout information)]]></stdout>
  <stderr><![CDATA[Using grid catalog type: lfc (standard stderr information)]]></stderr>
  <executionHosts>[ng-maat-server9.maatg.eu:8443/cream-pbs-neugrid,
 ng-maat-server8.maatg.eu]</executionHosts>
</GridJobStatus>
```

In this research, Pegasus (Deelman et al. 2005) has been selected over CRISTAL as a workflow management system based on several reasons. Firstly, it provides support for all workflow management operations and it has its own submission mechanism that can be integrated with Grid as well as a cluster (of Condor machines). Moreover, there are a few studies of workflow submission and execution on the Cloud - using a virtual cluster of Condor instances on virtual machines - using Pegasus (Vöckler et al. 2011). This provides a reference execution environment on the Cloud using Pegasus, which has been used for the purpose of the prototype. Furthermore, as discussed in Section 2.5.5, it also provides some information about the host machine (i.e. *Where* of WP7 model) along with name of the OS kernel (i.e. *Which* of WP7 model) on which a job has been executed. Using the information about the host, this research study has devised a mechanism to establish a mapping between the job and the Cloud resource used to execute that job, and use this mapping to re-provision similar resources on Cloud for reproducing a workflow execution.

The analysis used in N4U requires neuroimaging algorithms and executables and an environment already configured on the infrastructure. However, for the purpose of this research study such an environment was not available. In order to overcome this hurdle, a virtual image has been prepared that can support the execution of ReconAll workflow used in N4U. Moreover, this research work has borrowed the concepts of a scientific workflow from ReconAll and other domains to construct a simulated sample workflow that operates on input data and arguments to behave like a similar workflow used in N4U or in computer science. These workflows are then executed on the Cloud environment to verify this research hypothesis. The following sections provide details of the workflow execution scenario on the Cloud and the proposed architecture, named ReCAP, by keeping in mind the N4U requirements and the workflow reproducibility requirements, especially on the Cloud.

3.6 Workflow Execution Scenario on the Cloud

There have been projects (e.g. Williamson 2014; CERNVM 2015) that uses the Cloud for the execution of workflows. Mainly, these approaches create a virtual environment i.e. a virtual Grid on top of the Cloud resources using their legacy systems and execute workflows. A system, AMOS (Strijkers et al. 2010), presents a layer on top of a workflow management system and dynamically creates resources on the Cloud to instantiate a transient Grid ready for immediate use in the Cloud. A similar approach has also been discussed and tested by (Vöckler et al. 2011; Juve et al. 2009). It uses Pegasus as a Workflow Management System (WMS) along with the Condor (Tannenbaum et al. 2002) cluster on the Cloud infrastructure to execute workflow jobs. In this section, a scenario (see Figure 3.3) is presented that can be used to execute a workflow on the Cloud.

A scientist creates a workflow using a workflow authoring tool or uses an existing workflow from the Pegasus Provenance Store e.g. database and submits it to the Cloud infrastructure through Pegasus. Pegasus interacts with a cluster of compute resources in the form of Condor instances running on virtual machines (VM) in the Cloud. Each VM has a Condor instance to execute the user's job.

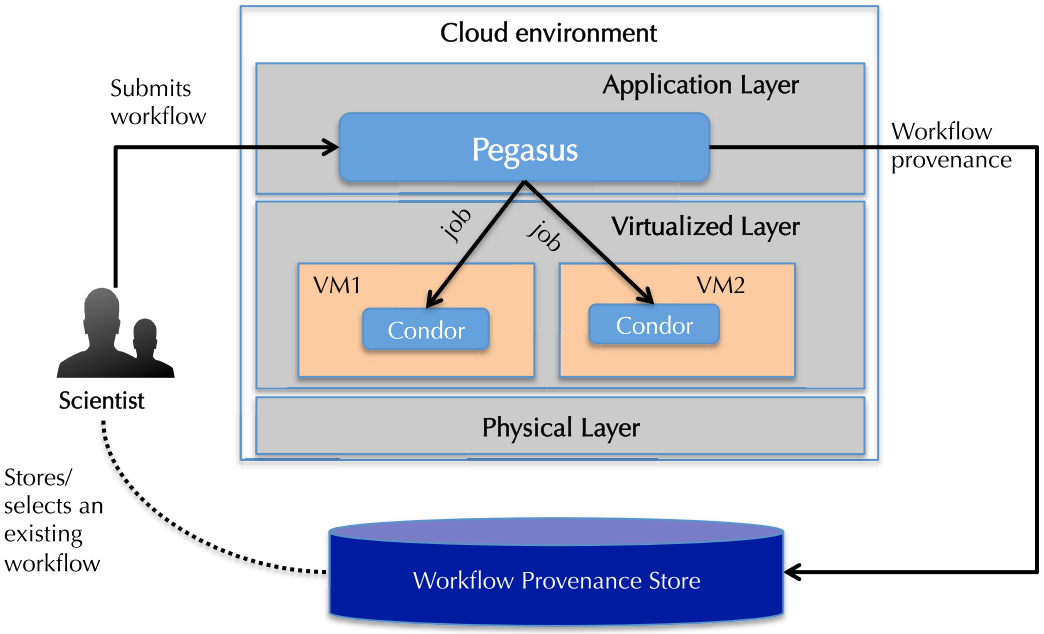


Figure 3.3: Workflow Execution on Cloud

Pegasus schedules the workflow jobs to these Condor instances and retrieves the workflow provenance information supported by the Pegasus database. The collected provenance information, which is stored in the Pegasus database, comprises job arguments (input and outputs), job logs (output and error) and host information. However, the collected host information is not sufficient to re-provision resources on the Cloud because Pegasus was designed initially for the Grid environment,

and such systems lack this capability at the moment (as discussed in Section 2.8). This workflow execution scenario on Cloud has inspired the general architecture of the proposed system discussed in following Section 3.7.

3.7 ReCAP: Architecture Overview

As mentioned before, the proposed ReCAP architecture is inspired by the mechanism used in (Vöckler et al. 2011) for executing workflows on the Cloud. Figure 3.4 illustrates the proposed architecture that is used to capture the Cloud infrastructure information and to interlink it with the workflow provenance collected from a workflow management system such as Pegasus. This augmented or extended provenance information compromising of workflow provenance and the Cloud infrastructure information is termed CAP. The components of this architecture are briefly explained below.

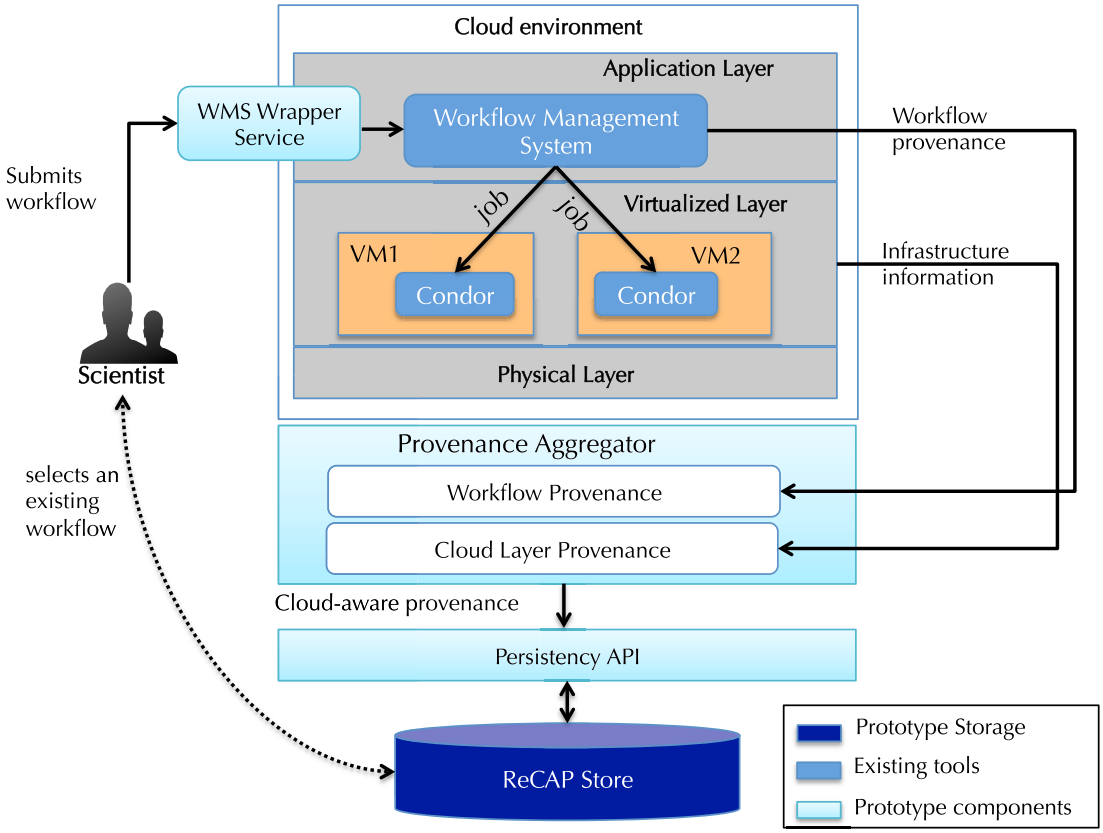


Figure 3.4: An abstract architecture of the proposed system (ReCAP)

- **WMS Wrapper Service**: This component provides a wrapper on top of an underlining WMS. It is responsible for receiving various user and CAP requests in submitting a user provided workflow and monitoring its status. As of now, there is no suitable online facility available that a user can use to submit a workflow and its associated files to Pegasus. Traditionally, a

command-based approach is used in which Pegasus provided calls are invoked from a terminal. With such a service based component, a user can submit his workflow through an HyperText Transfer Protocol (HTTP) client. Another purpose of this component is to engage with a user from the very first step of workflow execution i.e. workflow submission. Although this research study is focusing on workflow execution (discussed in Section 2.1.1), it still needs a mechanism to access the submitted workflow and its associated configuration files in order for it to reproduce and resubmit the same workflow. Therefore, such a mechanism was required that can act as an entry point for the system and also help in ensuring the access to the workflow source, which is one of the points of the reproducibility framework design for the Cloud (discussed in Section 3.3). Since the prototype is designed using a plugin approach, this gives the flexibility to introduce new plugins and thus provides support for more systems. For this prototype implementation, a Pegasus-based wrapper service has been implemented.

- **Workflow Provenance**: This component is responsible for receiving provenance captured at the application level by the workflow management system (e.g. Pegasus). Since workflow management systems may vary, a plugin-based approach is used for this component. Common interfaces are designed to develop plugins for different workflow management systems. The plugin also translates the workflow provenance according to the representation that is used to interlink the workflow provenance with the information coming from the Cloud infrastructure.

- **Cloud Layer Provenance**: This component is responsible for capturing information collected from different layers of the Cloud. To achieve re-provisioning of resources on the Cloud, this component focuses on the virtualization layer and retrieves information related to the Cloud infrastructure i.e. virtual machine configuration. This component is discussed further in Section 3.8.

- **Provenance Aggregator**: This is the main component task to collect and interlink the provenance coming from different layers as shown in Figure 1. It establishes interlinking connections between the workflow provenance and the Cloud infrastructure information. The provenance information is then represented in a single format that could be stored in the ReCAP Store through the interfaces exposed by the Persistency API.

- **Persistency API**: This acts as a thin layer to expose the provenance storage capabilities to other components. Through its exposed interfaces, outside entities such as the Provenance Aggregator would interact with it to store the workflow provenance information. This approach gives flexibility to implement authentication or authorization in accessing the provenance store.

- **ReCAP Store**: This data store is designed to store workflows and their associated provenance. It stores the mapping between workflow jobs and the virtual resources in the Cloud infrastructure. It also keeps a record of the workflow and its related configuration files being used to submit a user analysis on the Cloud. This information is later retrieved to reproduce the

workflow execution. However, it does not support workflow evolution in its current design because this is not the focus of this research work.

3.8 ReCAP: Detailed Architecture

This section presents the detailed architecture of the proposed system. ReCAP has been designed on the configuration and plugin-based mechanism. With this mechanism, support for new workflow management systems, mapping algorithms etc. can be easily added without changing the core of the system. There are seven key components in this design. They are the (i) WMS Wrapper Service, (ii) WS Client, (iii) WMS Layer, (iv) Cloud Layer, (v) Aggregator, (vi) WF-Repeat, and (vii) Comparator. Each of these components can further have their sub-components which are also discussed in this section. Figure 3.5 shows the detailed architecture of ReCAP and mutual interaction among its components.

3.8.0.1 ReCAP Configuration

ReCAP is designed using a plugin based approach and this requires a set of configuration parameters to drive the overall system. Consequently, the key aspects of the ReCAP such as WMS components, mapping algorithms, persistence API that interacts with the workflow provenance, the ReCAP databases and the Cloud middleware are driven by the configuration parameters. These configurations (shown as *ReCAP configs* in Figure 3.5) are divided into five main sections as shown in Appendix A.2. These sections are discussed as follow:

1. **cloud_settings**: Provides a large number of parameters which are mainly used to access the Cloud middleware. The implemented classes accessing the Cloud middleware to retrieve a Cloud resource, a Virtual machine or a file stored on the Cloud, information establish connection with the Cloud middleware using these configured parameters. Since the ReCAP prototype is using Apache Libcloud[2] API, which can interact with various Cloud middlewares, these parameters can be changed to accommodate a new Cloud middleware without any change in the code. Another important parameter in this section is *MAPPING_TYPE*, which informs the ReCAP to load the appropriate mapping algorithm.

2. **db_settings**: Provides information about the parameters used to connect with the database of the workflow management system. Since the prototype's persistency layer is built on top of an SQLAlchemy framework, which provides access to many databases such as Oracle, Sqlite, MSSQL etc., changing a database would not require a change in the persistency layer. For this prototype, Pegasus database settings on MySQL database have been used.

[2]https://libcloud.apache.org/

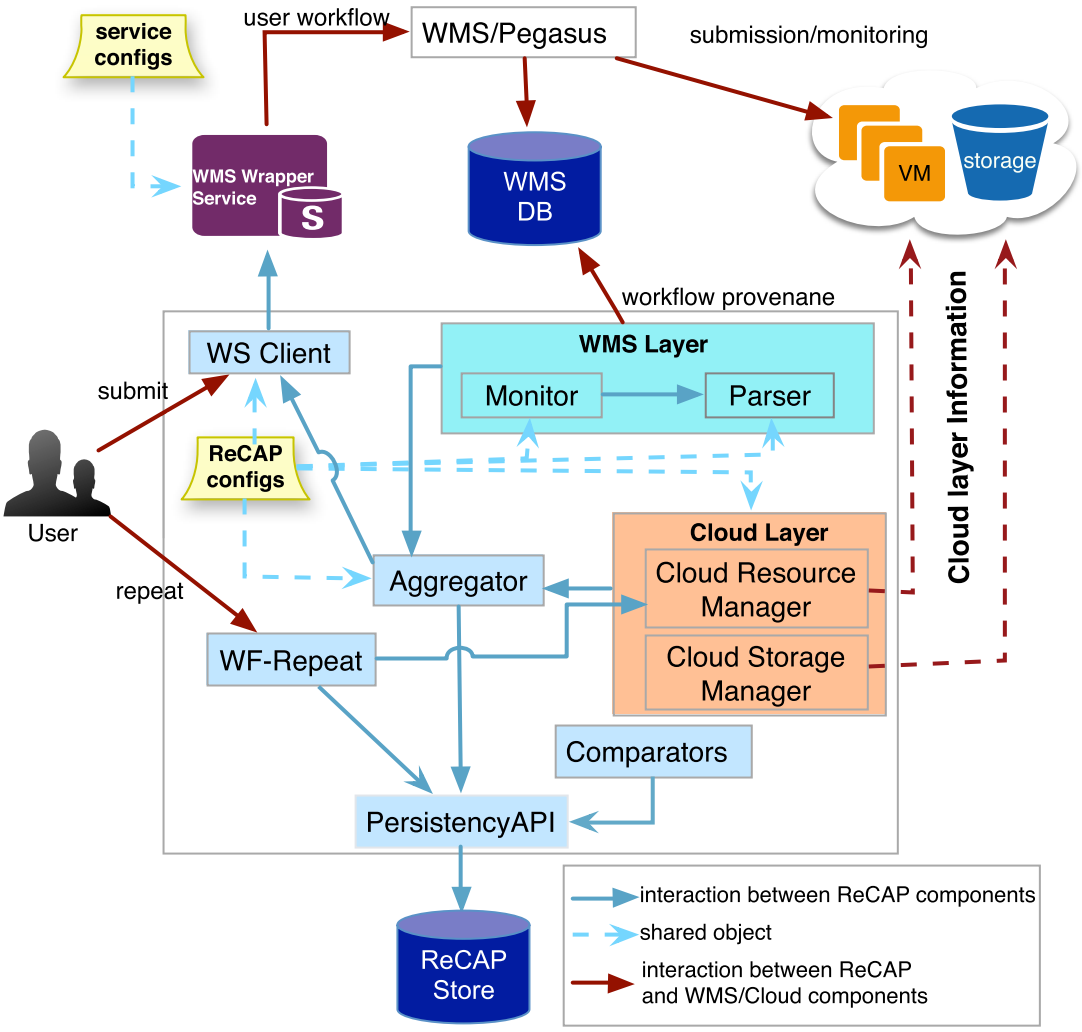


Figure 3.5: Detailed architecture of the ReCAP system

3. **cloudprov_settings**: Provides information about the parameters used to connect with the database used in the prototype. This database contains the relational schema shown in Section 3.8.9 and holds the mapping information between a job and a Cloud resource.

4. **WMS_settings**: Depending upon the used workflow management system, these parameters can be changed. For instance, *wms_monitor* parameter loads the appropriate monitoring component that monitors the workflow state in the database configured in above settings. This component is WMS specific and so is its implementation.

5. **WrapperService**: To interact with the WrapperService, the Client component requires connection information such as service URL and user credentials. This section provides these details.

6. **log_settings**: In order to log the inner activities of ReCAP, logging is provided and it is controlled by this parameter.

3.8.1 WMS Wrapper Service

This component of ReCAP is a RESTful web service that operates on top of a workflow management system, which for this study is Pegasus. It exposes interfaces through which a user can interact with the underlining workflow management system and can submit his workflows. As this service mainly interacts with a WMS, it resides on the same machine on which that system is running. For instance, the machine on which Pegasus runs and uses to submit workflows is called the Submit Host. In our case, Pegasus uses the Condor pool to execute workflow jobs. In this environment, the $SubmitHost$ is configured as the Master node for the Condor pool and all the VMs with the Condor instances acting as worker nodes. All these nodes create a Condor pool over the virtual machines which is termed a $VirtualCluster$ (Juve and Deelman 2010). Figure 3.6 illustrates the interaction of the WrapperService (WS) with other components of the system. In the case of Pegasus and Condor, as shown in Figure 3.6, this service interacts with Pegasus to submit the workflows received in the user's request. It also interacts if needed with the Condor pool through the $SubmitHost$. This interaction is useful for achieving Eager Resource-Job mapping (will be discussed later in 4.1.2).

A user accesses the Wrapper Services over HTTP and requests to submit his workflow. In this request, he will provide the abstract representation of his workflow i.e. DAX and its associated configurations, which are specific to the underlining workflow management system. In the case of Pegasus, he will provide a DAX file representing his workflow and a site file that provides information about storage, environment variables and workflow constraints. Upon receiving the request, the Service Wrapper firstly authenticates the user with the provided credentials in the service request. This enables the service to prevent unauthorized access to the underlying resources. At the moment, a very basic password-based HTTP authentication is implemented. However, it can be extended to database driven or more complex user authentication model.

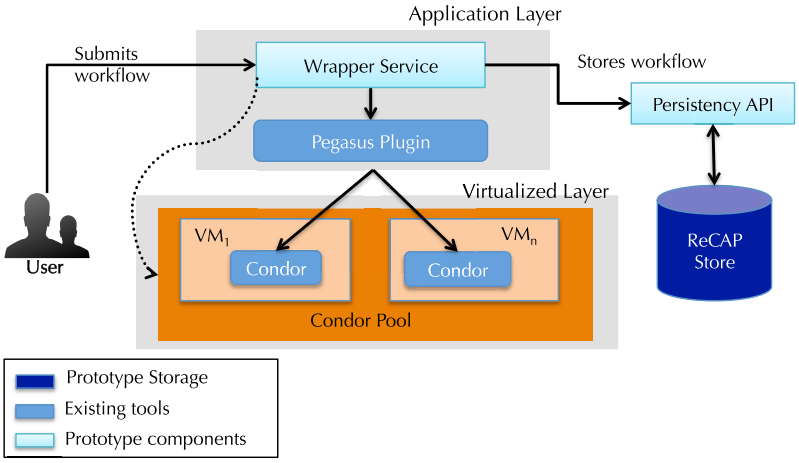


Figure 3.6: Interaction of Service Wrapper with overall system or its architecture

After successful user authentication, the Service Wrapper loads the plugin - specified in the service configuration (see Appendix A.1) - that can interact with the underlining workflow management system. As this study is using Pegasus, a Pegasus plugin is implemented that interacts with Pegasus commands and its database. The plugin stores the user provided files locally. The storage location on the local file system is controlled by a configuration parameter. After storing files, it then submits them to Pegasus using $pegasus-plan$[3]. This command parses the given abstract workflow into concrete workflow (DAG) and submits it to the Condor pool using Condor's DAGMan, which is responsible for managing DAG representation. An executable workflow, with all jobs and their dependencies, both control and data, is represented in the DAG representation. Once the workflow has submitted, Pegasus provides a unique identifier for this workflow and that is returned back to the user or a component as a response to its workflow submission request. This response is returned in JavaScript Object Notation (JSON) format, which is compact, flexible (schemaless) and easier to use. Upon successful submission of the user's workflow, the Service Wrapper stores the provided files in the database for later use in reproducing a workflow execution (will be discussed later in 4.3). The Figure 3.7 illustrates the flow of activities within the Service Wrapper for submitting a user workflow.

The following main operations are supported at the time of writing, which can be called from the service Client component.

- **submit**: This operation is used to submit a workflow to the underlying workflow management system, which is Pegasus in this prototype. All required files are passed in the service request, which are processed accordingly.
- **wms_get_file**: This operation is used to retrieve files (job outputs, workflow submission output) stored within the workflow management system's work directory. It takes two arguments

[3]pegasus-plan command http://pegasus.isi.edu/mapper/docs/4.0/cli-pegasus-plan.php

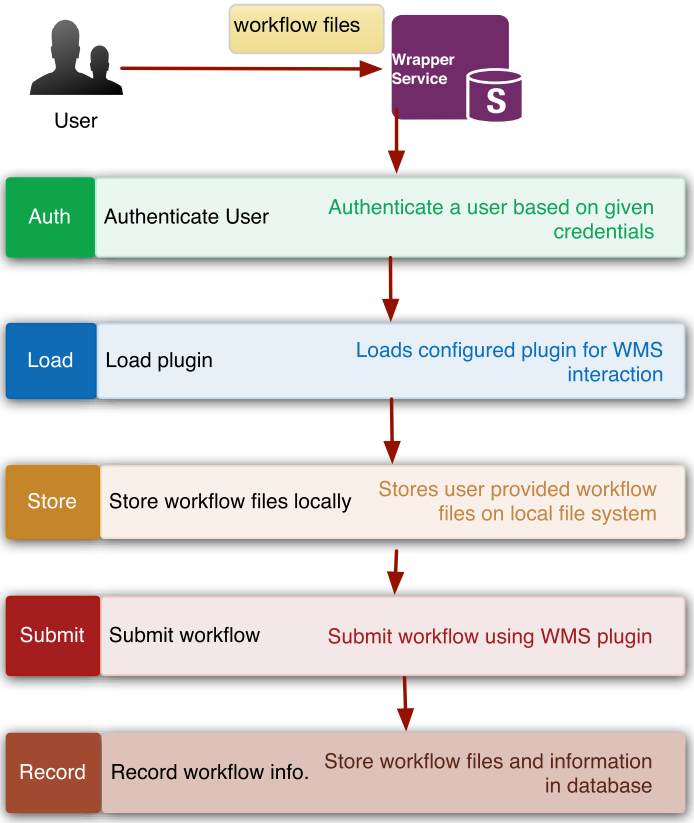


Figure 3.7: Flow chart of WMS Wrapper Service to process a workflow submission request

i.e. (1) job information and (2) the directory location to find the requested files. This operation is used while processing the job logs in mapping components.

- **jobmon**: This operation is Condor-specific as it attempts to retrieve the current status of the job running on Condor. It helps in retrieving the host information from the Condor pool on which a job is running. This operation is used in the Eager approach (discussed later in Section 4.1.2).

- **cpool_mips**: This operation is used to to retrieve the MIPS of the machines in the Condor pool. MIPS or KFLOPS are one way to specify the execution performance of a machine and it can affect a job execution performance (shown and discussed later in Section 6.2.3).

3.8.2 WS Client

In order to interact with the WMS-WS, the WS Client component of ReCAP is used. All interactions with the Wrapper Service pass through the WS Client component. On receiving requests from a user or other components such as the Monitor component of ReCAP, the Client component starts an HTTP session with the WrapperService. As authentication has been implemented in the WrapperService to avoid malicious access, it also provides user credentials along with the request. These settings are retrieved from the ReCAP configurations (see *WrapperService* in Appendix A.2) discussed earlier in Section 3.8.0.1.

In order to submit a workflow, the user interacts with the Client component and passes all required files. The Client component interacts with the WrapperService and submits the files. It retrieves the response from the WrapperService and uses the configured WMS Parser to parse it to extract the workflow ID and ReCAP ID assigned to it. It then starts the Monitor component to start monitoring the provenance information of the submitted workflow. This interaction is shown in Figure 3.8.

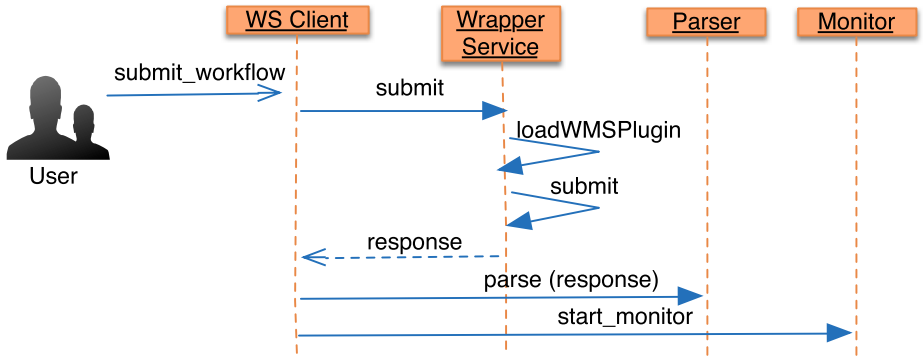


Figure 3.8: Illustrating the interaction between ReCAP components for submitting a user workflow

3.8.3 WMS Layer

As discussed earlier, the design philosophy of ReCAP is plugin-based in order to provide support for extensibility. A literature review has shown that multiple workflow management systems such as Pegasus, Chimera, Taverna and Kepler etc., provide either the hostname or the IP of the machine on which a job was executed. Therefore, it has been considered to enable support for multiple workflow management systems by adopting a plugin-based design. This section provides detail about the WMS Monitor and Parser components. During mapping process, this layer loads appropriate plugin implementation based upon the configuration parameters (see WMS_settings section in Appendix A.2). In this prototype, Pegasus-based plugins have been developed and tested.

3.8.3.1 Monitor

This component has been designed to monitor a workflow execution and retrieving its provenance information. This component presents a threaded implementation which enables continuous monitoring of a workflow execution. In order to perform monitoring operations, it interacts with the WMS database and retrieves workflow and job states. Once a workflow is finished, it starts the job to Cloud resource mapping operation using the configured mapper plugin. It also interacts with the Parser plugin to parse job outputs. A monitor plugin for Pegasus has been written in this prototype.

3.8.3.2 Parser

This component helps in parsing the files and job outputs produced during a workflow execution. Since each WMS can produce its own files during workflow execution and the job output formats can also be WMS specific, this component helps in providing an abstraction layer on top. For example, once a workflow is submitted through Pegasus, a submit file is produced that contains information about the output directory and workflow identifiers. Moreover, job output logs contains information about the location of the input and output files on the Cloud. These parsers can also be extended to extract the CPU spec from the job outputs if they cannot be retrieved from the workflow provenance database. All this information is important to feed the Monitor component and also to acquire provenance information about the consumed or produced output files on the Cloud, which is later used in workflow output comparison algorithm (discussed later in Section 4.2.3). ReCAP loads the Parser plugin using the global configuration. For instance, to parse Pegasus outputs, it loads the PegasusParser plugin.

3.8.4 Cloud Layer

To interact with the Cloud middleware, a component Cloud Layer named 'CloudLayerComponent' has been developed. It provides two types of interactions with the Cloud i.e. (a) to retrieve information about the virtual machines and also (b) to retrieve information about the workflow's input and output files stored on the Cloud storage service. These two types of interactions are handled by its two sub-components: the Cloud Resource Manager (CRM) and the Cloud Storage Manager (CSM). These two components are briefly discussed in following sections.

3.8.4.1 Cloud Resource Manager (CRM)

The CRM interacts with the Cloud IaaS service such as the *nova* service of OpenStack to manage virtual resources on the Cloud. The management involves operations such as retrieving resource information of virtual machines and the provisioning of new resources on the Cloud upon receiving new resource requests. The retrieved information about the currently running virtual machines include their metadata, OS images used in those VMs, flavours configuration used in VMs. This interaction is shown in Figure 3.9.

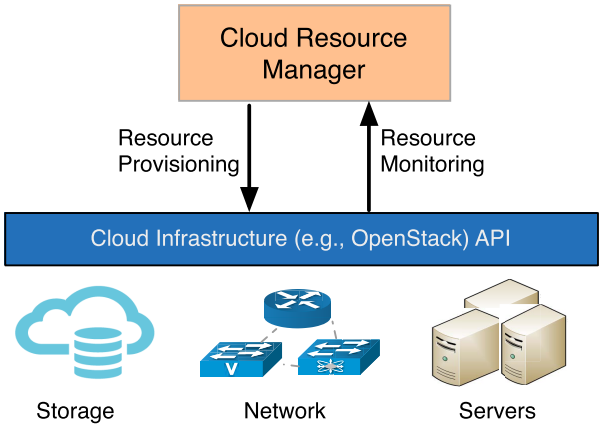


Figure 3.9: Interaction of CRM with the underlying Cloud infrastructure through APIs

3.8.4.2 Cloud Storage Manager (CSM)

This component interacts with the Cloud storage service such as Swift for the OpenStack Cloud middleware. As discussed earlier, the execution environment uses the Cloud storage service to save workflow inputs and outputs. Through this component, ReCAP is able to retrieve the files and their metadata from the Cloud. While storing the Cloud-aware provenance in the database, the Aggregator component invokes this component to retrieve filenames and their metadata such as MD5 hash and creation time etc. from a given location on the Cloud storage service. This component is

also used to iterate over the produced files on the Cloud during the workflow output comparison (discussed in Section 4.2.3).

3.8.5 Aggregator

The Aggregator (or also named Provenance Aggregator) component performs the mapping between the workflow job information collected from the Workflow Provenance component and the cloud resource information collected from the Cloud Layer Provenance component. In order to establish a mapping, it loads an appropriate mapping algorithm and this is driven by a configuration parameter *MAPPING_TYPE* (as discussed above). The mapping information is then stored in the database. Section 4.1 explains the mapping algorithms designed in this prototype.

3.8.6 WF-Repeat

This component is designed to re-execute a workflow on the Cloud. A user can select a previous workflow, identified with a unique ID, and request this component to re-execute it. Upon receiving the request, this component retrieves the required workflow source files from the ReCAP database and also Cloud-aware provenance information. Using this Cloud-aware provenance information, it re-provisions the resources on the Cloud infrastructure and then submits the workflow over them using the underlying workflow management system, which is Pegasus in this research. This process has been discussed in detail in Section 4.3.

3.8.7 Comparator

The comparator component performs provenance comparison operations (as discussed in Section 3.3) for evaluating the workflow reproducibility. The comparisons include a Workflow Output Comparison (discussed in Section 4.2.3) and a Workflow Graph Structure Comparison (discussed in Section 4.2.1) etc. It also performs provenance completeness and correctness analysis (discussed in Section 4.2.2) on the given workflow provenance traces. It takes two workflow identifiers to retrieve their provenance information from the database and then invokes the appropriate comparator implementation such as Workflow Output Comparison.

3.8.8 Persistency API

This acts as a thin layer to expose the provenance storage capabilities to other components. In order to interact with the underlying databases, a persistency layer is designed to provide a common callable

interface for multiple backend engine such as MySQL[4]. This API uses SQLAlchemy[5] that provides access to multiple database engines. An instance to the database connection can be obtained by specifying the connection parameters in the config file (see *wmsdb_settings* and *recapdb_settings* in Appendix A.2).

3.8.9 ReCAP database schema

A relational database schema has been designed that assists in storing workflow descriptions and their configuration files, workflow job-to-Cloud resource mappings, temporary mappings for the *Lazy* approach or the $SNoHi$ approach, and files metadata (consumed or produced data by the workflow jobs) stored on the Cloud. This information later helps in retrieving CAP information and also helps in answering Cloud-aware provenance queries and comparing workflow outputs. Following paragraphs describe the schema shown in Figure 3.10.

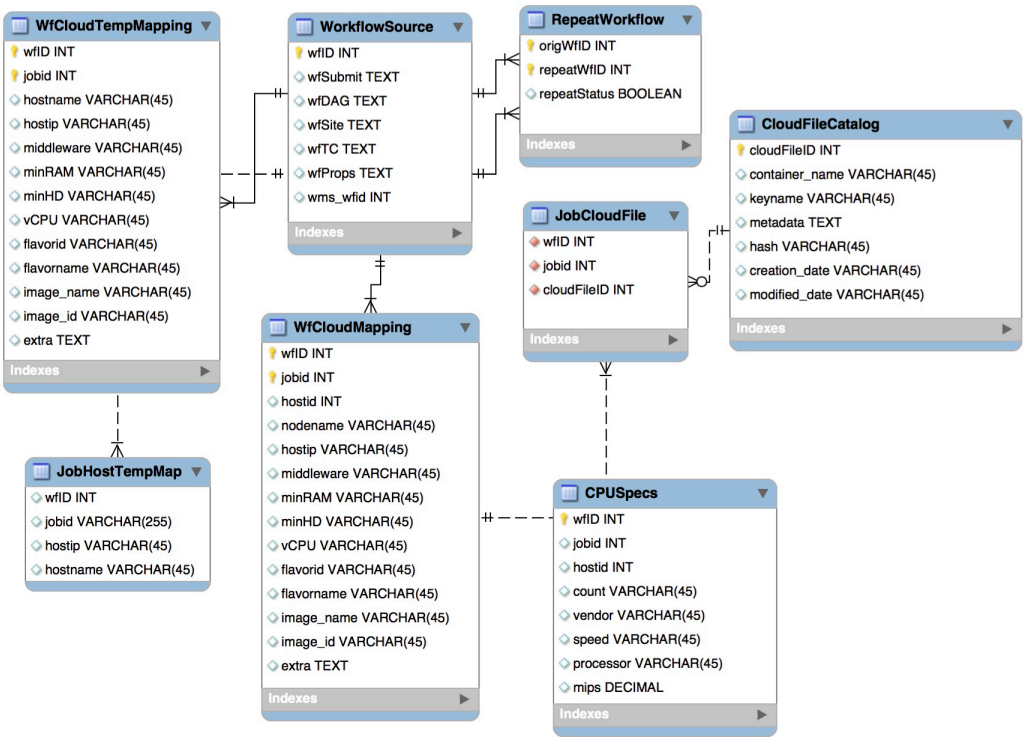


Figure 3.10: ReCAP Relational Database Schema

In order to preserve the original workflow and its associated configuration files, the *Workflow-Source* table is used. The *wfDAG* column stores the workflow representation described in a DAG format. This is an abstract workflow which will be submitted through Pegasus for execution. The *wfSite* column stores the information related to execution site and its storage elements. This information specifies the storage locations and paths to be used for reading and writing data during the

[4]https://www.mysql.com/
[5]http://www.sqlalchemy.org/

workflow execution. The *wfTC* column stores information about the executables which are to be used for workflow jobs. A user can also provide configuration properties to Pegasus that affects the way Pegasus plans, schedules and stores workflows. This set of properties is stored in the *wfProps* column. Each workflow is assigned a unique ID by Pegasus and also in the ReCAP database, thus a mapping between these two IDs are required. The *wfID* is the ID assigned to a workflow execution by ReCAP and the *wms_wfid* is the unique ID assigned to a workflow by the workflow management system such as Pegasus. The *wms_wfid* is essential to record because it is used in retrieving workflow provenance from the Pegasus. Since the prototype is designed using the Pegasus WMS, this table contains a few Pegasus specific entries. However, the same concepts can also be applied when used with other similar workflow management systems.

In order to establish the final job-to-Cloud resource mapping and to record Cloud-aware provenance information, the *WfCloudMapping* table is used. This table stores the mapping between workflow jobs and the configurations of the Cloud resources used for their execution. In order to specify the resource flavour, that provides the resource hardware configurations, *flavorname* and *flavorid* are stored. However, this information can be customised or new entries can be added by the Cloud Provider. For instance, Amazon EC2 provides an extensive list[6] of instance types with customized values. On the other hand, Openstack middleware used in Open Science Data Cloud (OSDC) provides a limited set of instance types (as shown in Figure A.1 in Appendix A). Therefore, the individual parameters i.e. *minRAM*, *minHD*, *minCPU* specifying a Cloud resource are also stored. As highlighted in Section 3.3, the information about the software stack is also essential to successfully conduct an execution. The *image_name* column provides the information about the operating system running the virtual machine. In this prototype, it is assumed that the operating system or the image contains all the required libraries on which a job executable is dependent. In order to store the billing information, which is also identified in Section 3.3, the *extra* column is stored provided the Cloud APIs support this functionality. This column basically stores data in a JSON format and thus enables us to store multiple key-value pairs in it. Due to this, it is also possible to store metadata information of a Cloud resource i.e. creation date, hash, etc. provided by the Cloud provider. This column has also been used to store the cost associated to the Cloud resource.

In order to tackle the dynamic resource scenario on the Cloud (discussed later in Section 4.1), the *WfCloudTempMapping* table is used. This table stores temporary mapping information which is then moved to the *WfCloudMapping* table once a workflow execution has finished. There is another table, *JobHostTempMap*, that stores temporary job and its execution host mapping. This table is used in the SNoHi mapping approach (discussed later in Section 4.1.4). In order to store more detailed information about the CPU, the *CPUSpecs* table is used. Although, the existing Cloud providers and their offerings do not allow a user to request a resource with such resource parameters, the impact of CPU performance still cannot be ignored especially for compute-intensive jobs. Moreover, this information is also helpful when sharing experimental setup and results with the peers. This is why

[6] http://aws.amazon.com/ec2/instance-types/

this information is captured and stored as part of the Cloud-aware provenance.

As discussed earlier, the data files are stored on the Cloud, this is why it is important to keep track of file locations and metadata on the Cloud. To achieve this, the *JobCloudFile* and *CloudFileCatalog* tables are conceived. The *JobCloudFile* table stores the mapping between workflow jobs and its produced/consumed files on the Cloud. The *CloudFileCatalog* provides detailed information about a file stored on the Cloud. This information includes the location of the file, specified by the *container_name* and *keyname*, MD5 *hash* of the file contents, additional *metadata* stored along with the file, *creation_date* and *modified_date*. By using this information, it is not only able to provide information about the file, but also can be helpful in comparing the file contents produced from workflow repeated executions to verify a workflow result. This also helps in identifying if a file is changed over time, which, however, is not the focus of this research study. For instance, a file created by a job will have the same creation and modification time or even no modification time. However, if a file is modified or tampered with somehow, the modification date will be updated. By comparing the latest dates with the already stored information, one can deduce if the file contents are changed.

3.9 Summary

This chapter discussed the computational reproducibility with a focus on preserving the computational environment, including both software and hardware. It also discussed the important factors in achieving reproducibility. Among the discussed factors, one key factor is the ability to execute an experiment on a similar execution environment, which is the main aim of this research work. Based on the analysed literature, Section 3.3 provided a set of requirements that could be used to achieve workflow reproducibility on the Cloud. A workflow can be reproduced if the execution environment, including hardware and software, is recreated along with workflow jobs and data. In the context of the Cloud, in which resources are provisioned on-demand, it is important to gather information about the execution infrastructure during workflow execution and this information should be made part of the collected provenance. To achieve this, the proposed solution should devise a mechanism that can collect such information and then interlink it with the existing workflow provenance. In this chapter, it is also discussed that provenance comparison mechanism can be used to verify workflow reproducibility. In doing so, the comparison algorithm should be able to compare provenance structures, provisioned resources along with workflow outputs.

In order to measure the correctness and completeness of the captured provenance information, key factors are discussed in the light of the literature (discussed in Section 3.4). Since the proposed approach does not rely on annotations, it employs the structural analysis of the provenance graph to deduce the completeness of the provenance graphs. By comparing the provenance graphs, it deduces the similarities between them. The correctness is determined by analysing the presence of

key parameters of a Cloud resource (identified in Section 3.3). This assists in answering the **Research Question 3** (discussed in Section 1.4).

This chapter then introduced the N4U community and the neuro-analyses being performed using workflows. It also discussed some of the related key requirements in N4U related to workflow execution and provenance. One such requirement is related to capturing workflow execution provenance. In N4U, workflows are executed on its Grid-based infrastructure, which is different from the Cloud environment. The services written for N4U to orchestrate a workflow execution are designed for the Grid-based environment, thus they cannot be used in this research work. Pegasus has been identified as a workflow management system that can be used in this research because Pegasus has already been used to submit workflows on Cloud by creating a virtual cluster of virtual machines (running Condor) on top of the Cloud infrastructure. This research study has also used the same execution model to submit workflow.

Based on the understanding of N4U provenance requirements and the Cloud environment, an architecture has been proposed and was discussed in Sections 3.7 and 3.8. The proposed architecture is designed in a plugin-oriented model, in which new systems i.e., workflow management systems or Cloud providers can be supported by developing new plugins and setting appropriate configuration values. The proposed architecture collects workflow provenance information from the workflow management system. It also collects the Cloud infrastructure layer information from the Cloud middleware using the Cloud APIs. These two sets of information are then combined together through the Aggregator component (discussed in Section 3.8.5) by using the mapping algorithms (discussed later in Section 4.1). In order to provide support for remote workflow submission, a plugin-based wrapper service is designed and implemented. For the purpose of this prototype, this service interacts with Pegasus and submits user provided workflows. In order to interact with Cloud services such as infrastructure and storage, separate components are designed which use a platform agnostic library such as Apache Libcloud to provide support for multiple Cloud providers.

The next chapter, Chapter 4 presents the implementation of the proposed system which will aim to capture and map job-to-resource information, and to reproduce a workflow execution on the Cloud by re-provisioning the same execution infrastructure. Chapter 4 discusses the Cloud resource usage scenarios and the appropriate mapping algorithms for the identified scenario. It also discusses the details of the WF-Repeat component (see Section 4.3) that helps in reproducing a workflow execution. It then discusses the provenance comparison approaches that are designed to deduce the extent to which two given provenance traces are similar or to say whether a workflow has been reproduced successfully.

Chapter 4

Workflow Reproducibility using Cloud-Aware Provenance (ReCAP) - Implementation

Based on the ReCAP architecture (discussed in Section 3.8), this chapter discusses the design and implementation of devised strategies to achieve a workflow job to the Cloud resource mapping, termed as job-to-Cloud resource mapping, in both static and dynamic Cloud environments. Once a job-to-Cloud resource mapping is established, this chapter then presents the mechanism for the re-execution of the workflows on the Cloud infrastructure using the Cloud-aware provenance information. It is then followed by a discussion on the workflow provenance comparison mechanism as identified in Section 3.3. The mechanism involves the comparison of the workflow provenance graph structure comparison, workflow outputs comparison and the Cloud infrastructure.

4.1 Job-to-Cloud Resource Mapping

This section discusses the job-to-Cloud resource mapping, which is later used for re-executing a workflow on similar Cloud resources using the mechanisms devised in this research study. Before diving into a detailed discussion of these mechanisms, first it is important to understand two different resource usage scenarios on the Cloud. These scenarios and their understanding provide a better picture of the requirements and the motivation behind devising different mechanisms to establish a job-to-Cloud resource mapping for each discussed scenario.

- **Static Environment on Cloud**

  In this environment, the virtual resources, once provisioned, remain in a RUNNING state on the Cloud for a longer time. This means that the resources will be accessible even after a workflow's execution is finished. This environment is similar to creating a virtual pool or Grid on top of Cloud's resources. A Static mapping scheme devised for such an environment will be discussed in Section 4.1.1

- **Dynamic Environment on Cloud**

  In this environment, VMs are shut down after the job is done. Therefore, a virtual resource, which was used to execute a job, will not be accessible once a job has finished. Moreover, in this environment, resources are provisioned on demand and released when they are no more required. Two approaches (a) Eager and (b) Lazy have been devised to handle this scenario and they will be discussed in Sections 4.1.2 and 4.1.3 respectively.

The mapping approaches discussed in the following sections achieve the job resource mapping using provenance information, which is available in many workflow management systems such as Pegasus or Chiron. One such information is an indication of the execution host or its IP in the collected provenance information, which is available across almost all systems. Many systems do maintain either name or IP information. In the Cloud's IaaS layer across one provider or for one user, no two machines can have same name or same IP at any given time. This means any running virtual machines should have unique IP. Once this virtual machine is destroyed, it is possible that the same IP is assigned later to a new virtual machine. All the rest of the properties of a virtual machine are accessible through IaaS layer and can be used by multiple machines at a time. For instance, multiple machines can be provisioned with flavour *m1.small* or with OS image Ububtu 14.04.

4.1.1 Static Approach

The CloudLayerProvenance component of ReCAP is designed in such a way that interacts with the Cloud infrastructure as an outside client to obtain the resource configuration information. As mentioned earlier in Section 4.1, this information is later used for re-provisioning the resources to provide a similar execution infrastructure in order to repeat a workflow execution. Once a workflow has been executed, Pegasus collects the provenance and stores it in its own internal database. Pegasus also stores the IP address of the virtual machine (VM) where the job is executed. However, it lacks other VM specifications such as RAM, CPUs, hard disk etc. The Provenance Aggregator component retrieves all the jobs of a workflow and their associated VM IP addresses from the Pegasus database. It then collects a list of virtual machines owned by a respective user from the Cloud middleware. Using the IP address, it establishes a mapping between the job and the resource configuration of the virtual machine used to execute the job. This information i.e. Cloud-aware provenance is then stored in the provenance store of ReCAP. The flowchart of this mechanism is presented in Figure 4.1.

In this flowchart, the variable wfJobs – representing a list of jobs of a given workflow – is retrieved from the Pegasus database. The variable vmList – representing a list of virtual machines in the Cloud infrastructure – is collected from the Cloud. A mapping between jobs and VMs is established by matching the IP addresses (see in Figure 4.1). Resource configuration parameters such as flavour and image are obtained once the mapping is established. The *flavour* defines the resource configuration

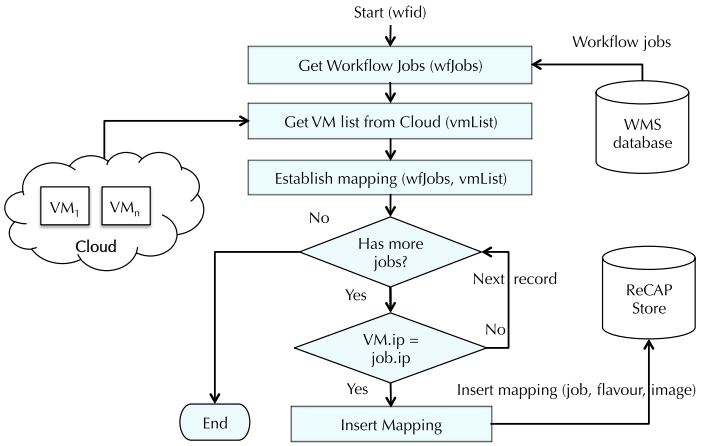


Figure 4.1: Flowchart of creating job-to-Cloud resource mapping using the Static approach

such as RAM, Hard disk and CPUs, and the *image* defines the operating system image used in that particular resource. By combining these two parameters together, one can provision a resource on the Cloud infrastructure. After retrieving these parameters and jobs, the mapping information is then stored in the ReCAP Store (see in Figure 4.1). This mapping information provides two pieces of important data the: (a) hardware configuration and (b) software configuration. As discussed in Section 3.3, these two parameters are important in re-provisioning a similar execution environment.

Algorithm 1 Job-to-Cloud resource mapping in Static Approach

Require: *wfJobs*: Set of jobs in the workflow.
vmList: Set of virtual machines in the Cloud infrastructure.

```
1:  procedure JobResourceMapping(wfJobs, vmList)
2:      cloudResources ← {}
3:      for all job ∈ wfJobs do
4:          for all vm ∈ vmList do
5:              if vm.ip = job.ip then
6:                  cloudResources[job] ← vm
7:              end if
8:          end for
9:      end for
10:     for all resource ∈ cloudResources do
11:         job ← resource.job
12:         resourceFlavor ← resource.flavor
13:         resourceImage ← resource.image
14:         insertToReCAPStore(job, resourceFlavor, resourceImage)
15:     end for
16: end procedure
```

Algorithm 1 presents the pseudo-code of the Static mapping approach. As discussed previously, this approach cannot work for a dynamic scenario in the Cloud as it establishes the mapping once

a workflow has finished and assumes that the machines on which the jobs are executed are still available and accessible. However, in the dynamic situation, Cloud resources will not be available once a job finishes its execution and it will not be possible to retrieve their resource configurations from the Cloud. To overcome this challenging scenario, the following mechanisms *Eager* and *Lazy* are devised which are capable of establishing a job-to-Cloud resource mapping for the Dynamic scenario. These approaches have been discussed in the following sections 4.1.2 and 4.1.3.

4.1.2 Eager Approach

The Eager approach has been devised to establish a job-to-Cloud resource mapping for the dynamic environment on Cloud. In this scenario, a resource may no longer exist on the Cloud when a job has finished. In that case, the information about its configuration cannot be retrieved from the Cloud middleware and the job-to-Cloud resource cannot be established. In this approach, the job-to-Cloud resource mapping is achieved in two phases. In first phase, temporary mapping between the job and Cloud resource is established. It is called temporary mapping because the job is still in running phase and it is possible that it may be scheduled to a different machine in case of job failure. Therefore, this initial mapping is temporary. In the second phase, the final job-to-Cloud resource mapping is established by retrieving job information from the workflow provenance captured by the WMS, which is Pegasus in this research work.

4.1.2.1 Temporary Job-to-Cloud Resource Mapping

In this phase, the Eager approach monitors the underlying WMS database i.e. Pegasus for the implemented prototype. In Pegasus, along with the host name, its database also maintains the job ID assigned to each job by Condor. The monitoring thread retrieves the condor id assigned to the workflow job and contacts the WMS Wrapper Service (WMS-WS) for information about the job. As the WMS-WS works on top of the underlying workflow management system, therefore it also has an access to the Condor cluster. Upon receiving the request, WMS-WS retrieves job information from the Condor. This information contains the machine IP on which the job is running. Based on this information, the CRM component retrieves the virtual machine information from the Cloud middleware based on the machine IP (as discussed in the Static Approach) and stores this information in the database. This information is treated as temporary because the job is not finished yet and there is a possibility that a job may be re-scheduled to some other machine due to runtime failures. The flowchart of this mechanism is presented in Figure 4.2.

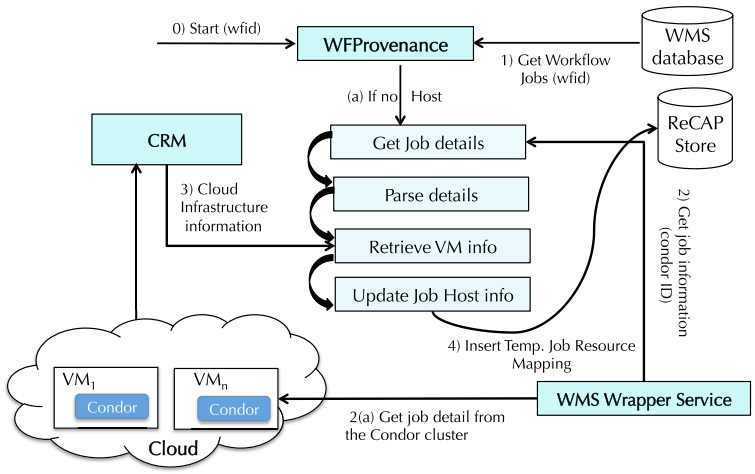


Figure 4.2: Flowchart to create temporary job-to-Cloud resource mapping using the Dynamic approach

4.1.2.2 Final Job-to-Cloud Resource Mapping

This phase starts when the workflow execution is finished. The Provenance Aggregator component starts the job-to-resource mapping process. In doing so, it retrieves the list of workflow jobs from the database and list of virtual machines from the Cloud middleware. It starts the mapping between the jobs and the virtual machines based on the IP information, stored in the database, associated with the jobs. In the case of not finding any host information in the database, which is possible in the Dynamic usecase, the Provenance Aggregator retrieves the resource information for that job from the temporary repository that was created in the first phase (as discussed in the above section). Upon finding the Cloud resource information, the Provenance Aggregator component registers this Cloud-aware provenance information in the ReCAP Store. Once the mapping for a job is established and stored in the database, its corresponding temporary mapping is removed in order to reduce the disk storage overhead. The flowchart of this mechanism is presented in Figure 4.3. The algorithm of Eager approach is shown in Algorithm 2.

4.1.3 Lazy Approach

The Eager approach is designed to deal with the dynamic Cloud environment. However, it relies upon underlying execution infrastructure such as Condor. To overcome these dependencies, another approach is devised to establish the job-to-Cloud resource mapping for the dynamic Cloud environment. This approach is named *Lazy*. This approach periodically accesses the Cloud and retrieves a list of virtual machines using a monitoring thread that monitors the current status of the available VMs running on the Cloud infrastructure. Each VM along with its creation time is iterated and stored in the ReCAP Store for later use i.e. in the job-to-Cloud resource mapping phase. This is named *Lazy* because it establishes job-to-Cloud resource mapping at the end of a workflow execution. Before, it does not maintain any temporary relation between a job and the virtual machine,

Algorithm 2 Job-to-Cloud resource mapping in Eager Approach

Require: $wfJobs$: Set of jobs in the workflow.

1: *Phase 1: Temporary job-to-Cloud resource mapping*
2: **procedure** JobMonitor($wfJobs$)
3: *vmList* ← getVMs
4: **for all** $job \in wfJobs$ **do**
5: *condorid* ← *job.condor* ▷ each job is assigned unique id
6: *ip* ← WSClient.getHostInfo*(condorid)*
7: *vm* ← $vmList[ip]$
8: **if** vm != None & not vmMappingExists(vm, job) **then**
9: resourceFlavor ← vm.flavor
10: resourceImage ← vm.image
11: createTempMapping($job, resourceFlavor, resourceImage$)
12: **end if**
13: **end for**
14: **end procedure**
15: *Phase 2: Final job-to-Cloud resource mapping*
16: **procedure** EstablishMapping($wfJobs$)
17: *vmList* ← getVMs
18: **for all** $job \in wfJobs$ **do**
19: **if** job.ip in vmList **then**
20: vm ← getVM(job.ip)
21: **else**
22: vm ← getTempJobMapping(job)
23: **end if**
24: **if** vm **then**
25: resourceImage ← vm.image
26: resourceFlavor ← vm.image
27: storeJobResourceMapping($job, resourceFlavor, resourceImage$)
28: removeTempMapping($job, resourceFlavor, resourceImage$)
29: **end if**
30: **end for**
31: **end procedure**

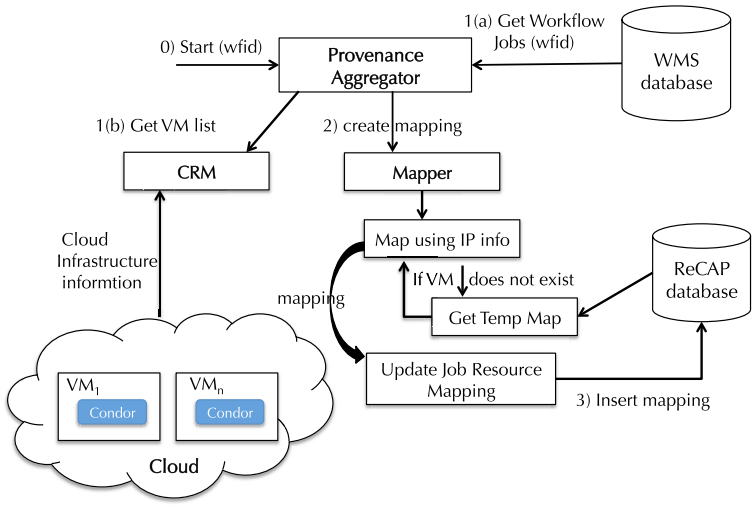


Figure 4.3: Establish the final mapping between Job and VM on Cloud

unlike the *Eager* approach that maintains a relation between a job and a resource during its phase 1. The algorithm of Lazy approach is presented in Algorithm 3.

This approach periodically monitors the available virtual machines on the Cloud infrastructure and retrieves their metadata information along with their creation time. This information is registered against the VM in a temporary table (see line 10). The database is updated only if new VM information is found on the Cloud (see line 7) against an already existing virtual machine. A new VM is determined mainly by its creation time. Once a job is finished, the mapping will be established using the Job's host information collected from the Pegasus database and the Cloud resources information from the ReCAP databases. The Cloud resource with the nearest start time from the job's own starts time and matching IP/hostname is selected as a resource for a given job. The advantageous and disadvantageous of this approach are given below.

- **Pros:** This approach may not be efficient in terms of discovery time, but it will work for all scenarios including the static environment because eventually it relies upon host information coming from Pegasus/WMS. It is also essential because during periodic monitoring this approach could not know which job is running one which machine.
- **Cons:** This approach will not be able to determine a mapping between a job and a virtual machine in case where no host information (due to a VM shutdown, job failure or no update in the WMS database) is available in Pegasus. As it does not maintain a temporary resource mapping, it cannot establish job-to-Cloud resource mapping in the absence of host information about a job.

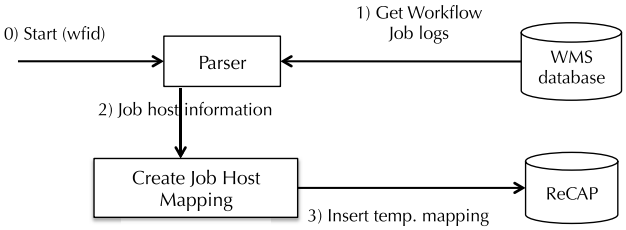
Algorithm 3 Job-to-Cloud resource mapping in Lazy Approach

Require: NIL

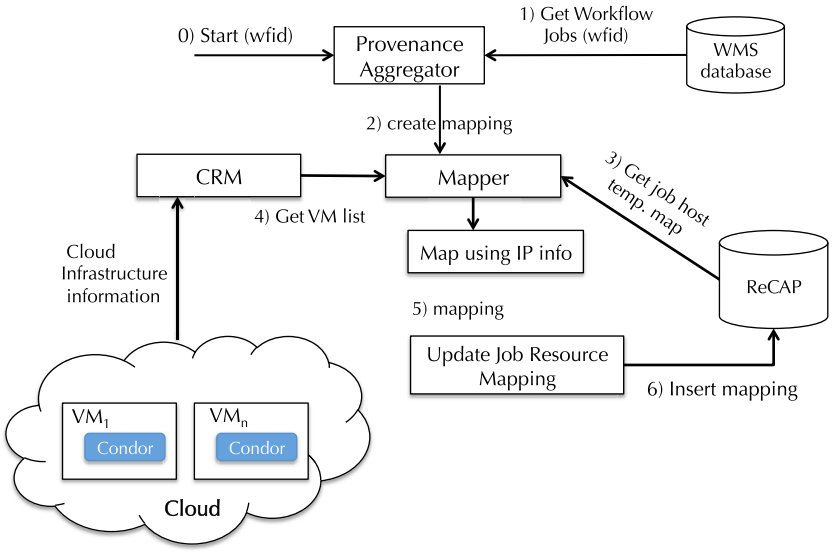
```
1: procedure MonitorCloudVirtualLayer(MONITOR_FLAG)
2:     while MONITOR_FLAG do
3:         vmList ← getVMs                                   ▷ Get a list of VMs
4:         for all vm ∈ vmList do
5:             vm_info ← vm.details
6:             vm_createtime ← vm.creation
7:             if not findVM(vm_info, vm_creationtime) then   ▷ check for new VM
8:                 resourceFlavor ← vm.flavor
9:                 resourceImage ← vm.image
10:                insertTempMapping(vm, resourceFlavor, resourceImage)
11:            end if
12:        end for
13:    end while
14: end procedure
15: procedure EstablishMapping(wfJobs) It is called when a workflow execution is completed
16:    for all job ∈ wfJobs do
17:        if job.ip! = None then
18:            ip ← job.ip
19:            vm ← getCloudVM(ip, job.start_time)   ▷ Get VM for given IP and creation
   time
20:            if vm != None then
21:                resourceFlavor ← vm.flavor
22:                resourceImage ← vm.image
23:                insertToReCAPStore(job, resourceFlavor, resourceImage)
24:            end if
25:        end if
26:    end for
27: end procedure
```

4.1.4 SNoHi Mapping

The aforementioned job-to-Cloud resource mapping approaches rely on host information either from the workflow management system's provenance repository or using the underlying infrastructure such as Condor supported by the WMS. However, the proposed mapping approaches will not work if none of these parameters are available. For workflow management systems such as Chimera (Foster et al. 2002) which do not maintain IPs or machine names as part of the job information in their provenance stores, another mapping approach has been devised. This approach has been named *Systems with No Host information* (SNoHi) mapping. In this approach, modified job scripts are to be sent that can capture and log machine IP/name information. These jobs can log this information in their output logs if their schema does not permit storing this information in the database.



(a) Parsing job logs to create temporary job host record



(b) Creating final job to Cloud resource mapping

Figure 4.4: Creating Job to Cloud resource mapping using the SNoHi approach

As discussed earlier about the Parser component, for each WMS there will be a dedicated parser plugin. The Parser component will retrieve the job logs from the WMS database and parse the host information. The parsed host information contains the IP and hostname of the resource on which the job was executed. This information is then stored in a temporary table (JobHostTempMap shown in the schema) which maintains only the job-host mapping. At this stage, the Cloud resource

configuration information is not stored. Figure 4.4a illustrates this stage in the SNoHi mapping. Once the job and its host information is available, the SNoHi mapping then can perform the job-to-Cloud resource mapping (as shown in Figure 4.4b). The Provenance Aggregator component will retrieve the workflow jobs from the WMS database and initiate the mapping. The Mapper will retrieve a list of VMs currently available on the Cloud infrastructure. It will locate the host information from the JobHostTempMap table for workflow jobs and establish the mapping between the job and the Cloud resource. This mapping information is then stored in the ReCAP database. The SNoHi mapping algorithm is given in Algorithm 4.

Algorithm 4 SNoHi Mapping Algorithm for WMS that does not maintain the machine IP/Name

Require: $wfJobs$: jobs of a workflow.

```
1:  procedure JobResourceMapping(wfjobs)
2:      jobHostMap ← getJobHostTempMap(wfjobs)
3:      vmList ← getVMs
4:      cloudResource ← {}
5:      for all job ∈ jobHostMap do
6:          for all vm ∈ vmList do
7:              if vm.ip = job.hostip then
8:                  cloudResources[job] ← vm
9:              end if
10:         end for
11:     end for
12:     for all resource ∈ cloudResources do
13:         job ← resource.job
14:         resourceFlavor ← resource.flavor
15:         resourceImage ← resource.image
16:         insertToReCAPStore(job, resourceFlavor, resourceImage)
17:     end for
18: end procedure
```

The algorithm first retrieves the job-host map from the JobHostTempMap table for the given workflow jobs (see line 2). It then retrieves the list of available virtual machines from the Cloud IaaS layer (see line 3). Since the host information has IP of the host, this can be used to establish mapping between a job and a Cloud resource. Once the job-to-Cloud resource mapping is established, the temporary records are deleted from the JobHostTempMap table.

4.2 Provenance Comparison

Comparing provenance has been used as a mechanism to determine reproducibility. Missier et al. 2013 propose the use of a provenance graph comparison to determine if both graphs are similar or

different. Since scientific workflows typically model a dataflow with a structure resembling a directed acyclic graph (DAG), the provenance comparison approaches measure similarity between two given provenance traces using graphs (Starlinger et al. 2014a). This section discusses the mechanism used to compare the provenance information of two workflows i.e. the original and the reproduced workflows to measure how much a reproduced workflow resembles the original execution. This mechanism is used in the Comparator component of ReCAP. For this, two provenance graphs are compared at three levels pointed out below.

1. **Workflow Graph Structure Comparison:** In this comparison, only graph structures are compared for two given workflows executions (original and reproduced) and it is determined if they are similar or different.
2. **Workflow Infrastructure Comparison:** In this comparison, the provisioned Cloud resources to execute workflow jobs are compared for two given workflow executions. This comparison ensures that the used infrastructure is similar or not.
3. **Workflow Output Comparison:** In this comparison, the data produced by workflow jobs are compared. This ensures that the output produced as a result of workflow execution is similar or not.

4.2.1 Workflow Graph Structure Comparison

In order to carry out this operation, first we need to transform the workflow provenance with the Cloud-aware provenance information into a graph representation. This is a Directed Acyclic Graph (DAG) because there is no loop involved in the workflow. The used graph representation is explained in the following section.

Structure-based approaches to measure workflow similarity perform some type of comparison on the two workflow graphs (Starlinger et al. 2014b). These comparisons can be of many forms. It could just be based on DAG structure i.e. comparing vertices and edges, and it could also include retrospective provenance data such as runtime information and execution parameters (Starlinger et al. 2014a). Since ReCAP also captures infrastructure information, it should also be accommodated in the graph and used in graph comparison. Before discussing the provenance graph comparison, the following section first discusses the graph representation used in this work.

4.2.1.1 Graph Representation

The workflow provenance can be represented as $G =< V, E >$, where G is a workflow provenance graph and it consists of a set of vertices V and a set of edges E. In this thesis, both vertices and

edges are labelled. V is a set of vertices that can be defined as $V = N \cup H \cup F$ (where N is job nodes during workflow execution and H represents virtual machines where the job execution took place and F is the files/data produced or consumed during workflow execution). The job nodes also include the augmented jobs created at runtime by the workflow management system to support data stage-in, stage-out or clean-up operations. A sample graph representation is shown in Figure 4.5. In this figure, there are two different types of job nodes; a) workflow job (in Green) specified by the user, and b) augmented job (in magenta) added by the workflow management system e.g. Pegasus. In the analysis of provenance graph, both these job types are treated as job nodes.

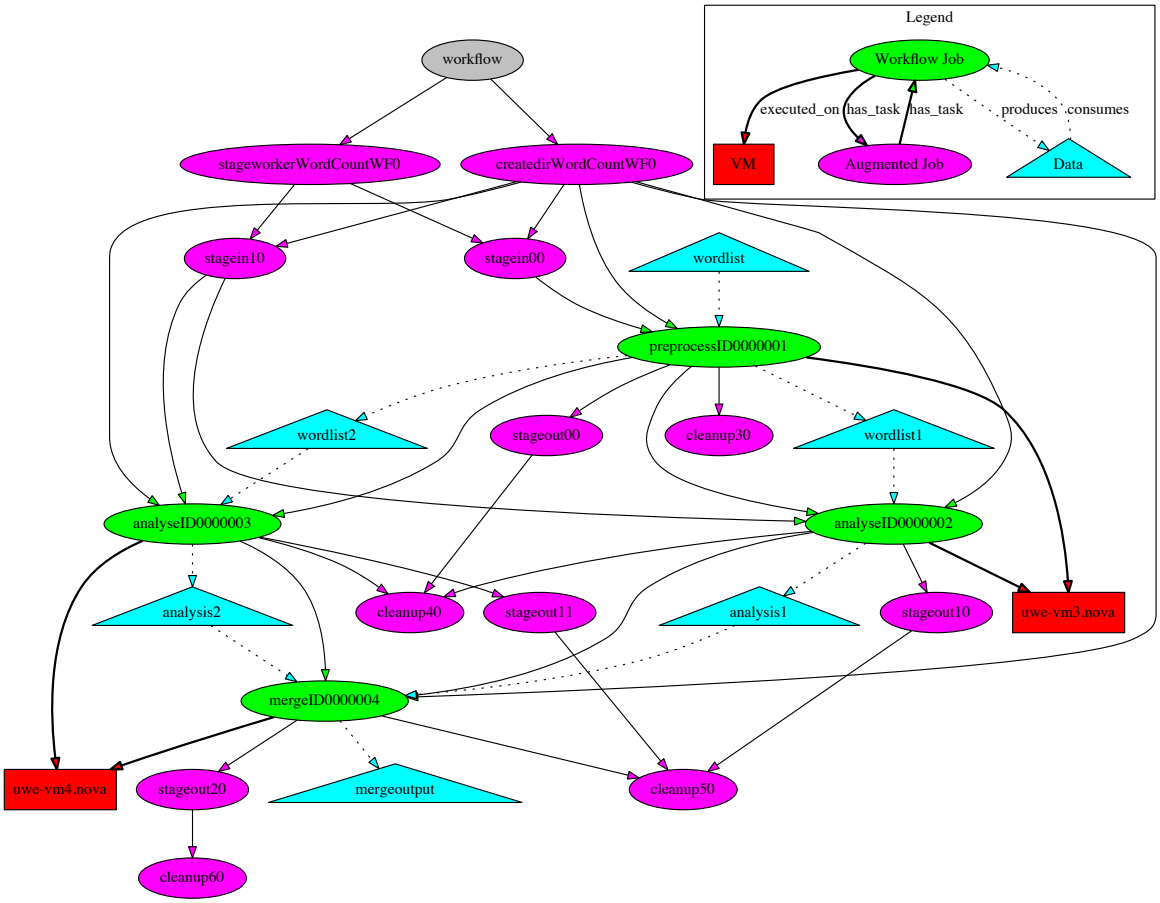


Figure 4.5: Provenance graph generated from a workflow execution

E is a set of edges and it is defined as

$$E = E_{task} \cup E_{exec} \cup E_{in} \cup E_{out}$$

where E_{task} represents the task edge between workflow jobs produced and executed by the

Pegasus. E_{exec} represents the edges between jobs and the hosts H on which the workflow job was executed. E_{in} represents the edges between input data and a job consuming the data. E_{out} represents the edges between output data and a job producing the data. In this graph, only connected nodes with at least one of the above mentioned edges are considered.

The following are the representation of different edges between two given nodes in the generated provenance graph G.

$$E_{task(i,j)} = N_i, E_{task}, N_j$$

$$E_{exec(i,h)} = N_i, E_{exec}, H_h$$

$$E_{in(i,f)} = N_i, E_{in}, F_f$$

$$E_{out(i,f)} = N_i, E_{out}, F_f$$

Based on this graph description, two provenance comparison algorithms have been devised. The first algorithm, *Provenance Graph Boolean Comparison* (see Section 4.2.1.2), performs the provenance graph comparison and determines the similarity between them. The outcome of this algorithm is True or False. This is why it is named *Provenance Graph Boolean Comparison*. However, it is possible that two graphs are similar but slightly varies in edges or nodes. In order to determine the quantifiable analysis of graph comparison, *Provenance Graph Numerical Comparison* has been discussed in Section 4.2.1.3.

4.2.1.2 Provenance Graph Boolean Comparison

The provenance graph Boolean comparison algorithm includes the following criteria.

1. Graph Size

   The graph size refers to counting the number of nodes/vertices. This similarity check is known as the isomorphic property of a graph. In this case, the graph size is counting the nodes including jobs, hosts, and data file along with the edge counts. Both given graphs should have same count.

   $$|WF_o| = |WF_r|$$

   Here WF_o refers to the original workflow provenance graph and WF_r refers to the provenance graph of the reproduced workflow.

2. Workflow Comparison with edges

   It is also important to compare the edges in the given two workflows. As discussed earlier, edges establish a relation between two vertices in a graph. Since the graph representation, shown in Figure 4.5, uses four different edges between three different nodes, it is important

to compare all these different edges in both given provenance graphs. For two provenance graphs to be determined as same, they both should have same nodes and same relationship i.e. edges between them. Following criteria is used to determine that the edges in both the given workflow provenance graphs are same.

$$\forall Edges_{task} \in WF_o, \sum_{i=1}^{n} E_{task_i} = \forall Edges_{task} \in WF_r, \sum_{i=1}^{n} E_{task_i}$$

$$\forall Edges_{exec} \in WF_o, \sum_{i=1}^{n} E_{exec_i} = \forall Edges_{exec} \in WF_r, \sum_{i=1}^{n} E_{exec_i}$$

$$\forall Edges_{in} \in WF_o, \sum_{i=1}^{n} E_{in_i} = \forall Edges_{in} \in WF_r, \sum_{i=1}^{n} E_{in_i}$$

$$\forall Edges_{out} \in WF_o, \sum_{i=1}^{n} E_{out_i} = \forall Edges_{in} \in WF_r, \sum_{i=1}^{n} E_{out_i}$$

A provenance graph structure comparison algorithm is presented in Algorithm 5. It traverses both the given workflow provenance traces (original execution and reproduced execution). It first checks whether both graphs have similar nodes, otherwise a difference is returned (see lines 2 and 3). After this, the graph traversal starts from the root node. For traversing a graph, a recursive function (termed the CompareNode) is implemented that traverses each graph vertex until no further child vertices are found. For each graph vertex, it collects its successors (i.e. children) and collects all edges for every node (lines 10-17). Since there are four types of edges, these edges are passed to a filtering function (termed the ProcessEdges) to determine the edge type. Each edge is added to an edge collection of that type. Later these edge collections (taskEdge on line 25 and consumeEdges on 27, etc.) are compared for both graphs and it is determined whether both graphs have same edges or not. For job-to-VM edge ($executed\_on$ edge), VM configurations i.e. hostinfo, including the RAM, CPUs, HD and flavour and OS image, are stored in the edge collection (see line 31). It is important to note that the VM configurations are considered when processing the graph node representing the VM. Therefore, name of the virtual machine is not significant for this algorithm.

4.2.1.3 Provenance Graph Numerical Comparison

Algorithm 5 can perform a graph structure comparison and provide an answer: True or False. However, it cannot inform the extent of similarity or dissimilarity between two given provenance traces. To achieve a deterministic or quantifiable answer for provenance graph structure comparison, the following graph comparison model is proposed.

Let there be two provenance graphs $PROV_o$ and $PROV_r$ representing original workflow execution and the reproduced workflow execution respectively. As discussed before, a graph is composed

Algorithm 5 Graph Structure Comparison Algorithm

Require: $PROV_o$: Original workflow provenance graph.
$PROV_r$: Reproduced workflow provenance graph.

```
1: procedure ProvGraphStructComp(PROV_o, PROV_r)
2:     if PROV_o.nodes ≠ PROV_r.nodes then
3:         return the different PROV_o.nodes − PROV_r.nodes
4:     end if
5:     Root_o ← PROV_o.root
6:     Root_r ← PROV_r.root
7:     CompareNodes(PROV_o, PROV_r, Root_o, Root_r)          ▷ Traverse the graphs
8: end procedure
9: procedure CompareNodes(PROV_o, PROV_r, n, ń)
10:    S_o ← PROV_o successors(n)
11:    S_r ← PROV_r.successors(ń)
12:    if len(S_o) = len(S_r) then
13:        for all c in S_o and ć in S_r do
14:            E ← PROV_o.edges(c)
15:            É ← PROV_r.edges(ć)
16:            ProcessEdges(E)
17:            ProcessEdges(É)
18:            CompareNodes(PROV_o, PROV_r, c, ć)          ▷ Recursive call
19:        end for
20:    end if
21: end procedure
22: procedure ProcessEdges(edges)
23:    for all e in edges do
24:        if e.type = task then
25:            taskEdges ← e
26:        else if e.type = consumes then
27:            consumeEdges ← e
28:        else if e.type = produces then
29:            producesEdges ← e
30:        else if e.type = executed_on then
31:            hostEdges ← hostinfo
32:        end if
33:    end for
34: end procedure
```

of vertices V and edges E. Let V_o be the number of vertices and E_o be the number of edges between the vertices of the $PROV_o$ graph. Let V_r be the number of vertices and E_r be the number of edges between the vertices of the $PROV_r$ graph. Since $PROV_r$ is to be compared against $PROV_o$, it is imperative to determine the number of common vertices and common edges between these two graphs. Let CV be the common number of vertices and CE be the common number of edges between two graphs. In order to calculate the CE, consider only those edges whose vertices that $\in CV$. As the graph structure is composed of vertices and edges, the following rule can be used to determine the numeric value for a provenance graph structure.

$$S = V + E \tag{4.1}$$

In case of the original workflow, the structure can be determined in terms of its number of nodes and number of edges as given below.

$$S_o = V_o + E_o \tag{4.2}$$

where V_o and E_o represent the number of vertices and edges of the base graph respectively. Since we also have a numeric value for common edges and vertices, the value for common graph structure (CS) can be calculated as follows:

$$CS = CV + CE \tag{4.3}$$

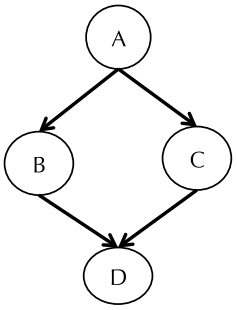
With these values known, the structural similarity (SS) of both the given provenance graphs can be determined as follows:

$$SS = CS/S_o \tag{4.4}$$

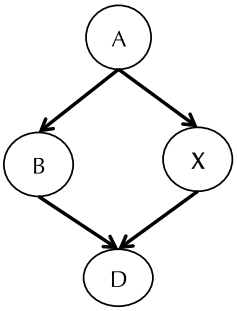
This structural similarity is calculated as a ratio to the given base graph. Since the reproduced workflow is compared against the original execution, S_o is used as a base graph. In order to understand this model, assume the sample provenance graphs shown in Figure 4.6 can be used. The graph G1 represents a provenance trace of original workflow execution. Let us assume that the other two graphs i.e. G2 and G3 are generated from the provenance information of the reproduced workflow. As can be seen all these graphs are slightly different to each other. If we apply the Algorithm 5 to compare G1 and G2, or G1 and G3 graphs, it will produce False on the comparison test. With the numeric model (discussed above), we can determine the quantifiable value for structural similarity.

Based on the base graph G1 of the original execution, the count for V_o is 4 and E_o is also 4. Therefore, the S_o value is 8. While comparing the graph G2 with G1, the common vertices are 3 because vertex X does not exist in graph G1. This influences the common edges count, which is 2 in this case, as well because the edges between A->X and X->D cannot be accommodated according to

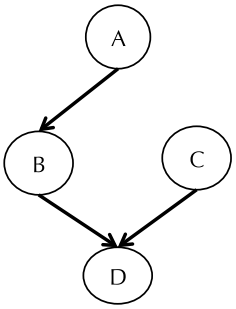
the model because these edges have one vertex that does not belong to CV. The SS value for graph G1 and G2 will be 0.625 which is 62.5% similarity. Similarly, for graph G3, the structural similarity (SS) with G1 is 0.875 which is 87.5 % similarity because it only misses one edge from A to C.



(a) Original Workflow provenance graph G1



(b) Example graph G2 to be compared with Original graph



(c) Another example graph G3 to be compared with Original graph

Figure 4.6: Example provenance graphs

The above calculated SS value will produce asymmetric results if the base graph is changed. For instance, if the base graph is G3, and G1 is compared with G3, the calculated value will be different from SS(G1, G3). With the above calculation, SS(G1, G3) is 87.5%, however SS(G3, G1) becomes 100%, which is an incorrect result. To avoid this situation, the ratios of the common vertices to the total vertices and the common edges to the total edges from both graphs should be calculated. In order to do this, the following equation is used. This equation aids in eliminating the asymmetric result which was present in the previous calculation. With this new equation, the values for SS(G1, G3) and SS(G3, G1) are both 92%.

$$SS = \frac{CV}{V_o + V_r} + \frac{CE}{E_o + E_r} \tag{4.5}$$

4.2.2 Correctness and Completeness Analysis of Cloud-Aware Provenance

This section presents the mechanisms to determine the completeness and correctness of the collected provenance. As discussed earlier in Section 3.4, the completeness of the collected provenance means

that information related to all of the jobs described in a workflow is correctly collected. Similarly, the correctness of the captured provenance has been defined as *"the extent to which the Cloud specific information is accurately acquired in the Cloud-aware provenance"*. Following subsections discusses the approaches devised to determine the correctness and completeness of the captured provenance information.

4.2.2.1 Correctness Analysis

In order to determine whether the captured provenance has correctly gathered the Cloud infrastructure information, this mechanism (shown in Algorithm 6) iterates over the jobs described in the workflow and their associated job-to-Cloud resource mapping information from the ReCAP database. For each job, it checks the presence of the identified Cloud parameters (see Section 3.3) i.e. configuration such as flavour and image information. The parameters include vCPU, RAM, Hard disk, flavour and image information. In case of missing any of these parameters, the algorithm treats this record as incorrect provenance record (see line 9). In case all the parameters are present in the job's mapping, the algorithm treats it as a correct provenance record (see line 7).

Algorithm 6 Check Correctness of the collected provenance

Require: $wfID$: ID assigned to a workflow (original) by the WMS.

```
1:  procedure CorrectnessCheck(wfID)
2:      wfJobInstances ← getWFJobInstances(wfID)        ▷ Get executed jobs by WMS
3:      jobCapMap ← getCAPInformation(wfJobInstances)   ▷ Get CAP mapping
4:      for all jobid ∈ wfJobInstances do
5:          mapping ← jobCapMap.get(jobid)
6:          if mapping.has("RAM") & mapping.has("vCPU") & mapping.has("HD") & mapping.has("flavour") & mapping.has("image") then
7:              Treat this provenance record as correct
8:          else
9:              Treat this provenance record as incorrect
10:         end if
11:     end for
12: end procedure
```

4.2.2.2 Completeness Analysis

As proposed in (Cheah and Plale 2012) and discussed in Section 3.4, the structural analysis on the Cloud-aware provenance graph is performed and compared against the actual workflow description in order to measure the completeness of the provenance information. As this comparison involves workflow description, the devised mechanism should have access to the actual workflow description.

Since ReCAP stores workflow description, it can compare the collected provenance with the workflow description. In this research study, the collected provenance also involves the information about the Cloud infrastructure that was used to execute a workflow. Therefore, the completeness analysis involves comparing the given workflow provenance with the actual workflow representation (in DAG format) along with the collected Cloud provenance and determines whether the ReCAP framework has collected provenance for all the jobs described in the workflow (as shown in Algorithm 7). This helps in answering **Research Question 3** (discussed in Section 1.4)

Algorithm 7 Check Completeness of the collected provenance

Require: $wfID$: ID assigned to a workflow (original) by the WMS.
$recapID$: ID assigned to same workflow in ReCAP mapping store

```
1:  procedure CompletenessCheck(wfID, recapID)
2:      wfDAG ← getWorkflowSource(wfID, recapID)
3:      wfActualJobs ← wfDAG.jobs          ▷ List of jobs declared in the workflow description
4:      wfJobInstances ← getWFJobInstances(wfID)          ▷ Get executed jobs by WMS
5:      jobCapMap ← getCAPInformation(wfJobInstances)          ▷ Get CAP mapping
6:      COMPLETENESS ← 0                                          ▷ initialize
7:      for all jobid ∈ wfJobInstances do
8:          mapExists ← jobCapMap.has(jobid)
9:          recapExists ← CheckRecapParams(jobCapMap[jobid])    ▷ Check Cloud parameters
10:         if mapExists and recapExists then
11:             COMPLETENESS ← COMPLETENESS + 1
12:         end if
13:     end for
14:     if len(wfJobInstances) < wfActualJobs OR COMPLETENESS < wfActualJobs then
15:         return {status: False, CScore: COMPLETENESS}          ▷ Completeness check Failed
16:     end if
17:     return {status: True, CScore: COMPLETENESS}          ▷ Completeness check passed
18: end procedure
```

In ReCAP, a workflow description is stored and assigned a unique ID ($recapID$) in the WorkflowSource table (as discussed previously in Section 3.8.9). In case of Pegasus, this workflow description is represented in DAG XML. This workflow description is retrieved from the database and parsed to get the actual jobs described in the workflow (see line 3). It then collects the retrospective provenance i.e. jobs executed by the WMS from the database (see line 4). At this point, the inter-job dependencies (represented as edges between job nodes) are not explicitly considered in the algorithm. This is due to the fact that each WMS executes workflow jobs according to an execution order specified in the workflow description and the information collected from the WMS contains this inherent inter-job dependency.

Along with this, it also retrieves the job-to-Cloud resource mapping information for all the jobs executed by the WMS (see line 5). Since this research is focusing on the Cloud infrastructure

provenance information, particular focus is given to the Cloud resource configuration parameters. After collecting the job-to-Cloud mapping, it iterates over the jobs and determines whether a given job has the correct Cloud infrastructure information. If a job-to-Cloud mapping exists for a job and all the Cloud parameters are correctly stored, it increments the COMPLETENESS counter. This counter is later used to determine the completeness status of the given workflow provenance. The completeness status evaluates to $False$ on two conditions (see line 14); (a) number of jobs executed by the WMS is lower than the actual number of jobs described in the workflow, and (b) COMPLETENESS counter is lower than the actual number of workflow jobs. The first scenario can occur if all jobs could not be executed by the WMS due to job failures. The second scenario can occur if either job-to-Cloud resource map does not exist for a job or the collected Cloud parameters are not correct (see line 10). In both these cases, the collected provenance is determined as not complete (see line 13) when compared with the workflow description. However, the collected provenance is treated as complete if there exists a correct job-to-Cloud resource mapping for all the jobs executed by the WMS and described in the workflow description.

4.2.3 Workflow Output Comparison

Another aspect of workflow repeatability is to verify that it has produced the same output that was produced in its earlier execution (as discussed in Section 3.3). In order to evaluate workflow reproducibility, an algorithm has been proposed that compares the outputs produced by two given workflows. It uses the MD5 hashing algorithm (Stallings 2010) on the outputs and compares the hash value to verify the produced outputs. The MD5 algorithm outputs a string that is often used in many systems as the single data identifier, due to a small collision probability (Lifschitz, Gomes and Rehen 2011). The two main reasons for using a hash function to verify the produced outputs are: a) simple to implement and b) the hash value changes with a single bit change in the file. If the hash values of two given files are same, this means that the given files contain the same content. Moreover, many systems such as Kepler also stores MD5 value of the data in order to provide data integrity and verify changes in data.

The proposed algorithm (as shown in Algorithm 8) operates over the two given workflows identified by srcWfID and destWfID, and compares their outputs. It first retrieves the list of jobs and their produced output files from the Provenance Store for each given workflow. It then iterates over the files and compares the source file, belonging to srcWfID, with the destination file, belonging to destWfID. Since the files are stored on the Cloud, the algorithm retrieves the files from the Cloud (see lines 11 and 12). Cloud storage services such as OpenStack Swift[1] and Amazon Storage Store (S3)[2] use the concept of a bucket or a container to store a file. This is why $src\_container$ and $dest\_container$ along with $src\_filename$ and $dest\_filename$ are given in the GetCloudFile

[1] http://swift.openstack.org
[2] https://aws.amazon.com/s3/

Algorithm 8 Compare Workflow Outputs Algorithm

Require: $srcWfID$: Source Workflow ID.
$destWfID$: Destination Workflow ID

```
1:  procedure CompareWorkflowOutputs(srcWfID, destWfID)
2:      srcWorkflowJobs ← getWorkflowJobs(srcWfID)
3:      destWorkflowJobs ← getWorkflowJobs(destWfID)
4:      FileCounter ← 0
5:      ComparisonCounter ← 0
6:      for all jobfiles ∈ srcWorkflowJobs do
7:          src_container ← jobfiles.container_name
8:          src_filename ← jobfiles.file_name
9:          dest_container ← destWorkflowJobs[jobfiles.jobname]
10:         dest_filename← destWorkflowJobs[jobname].file_name
11:         src_cloud_file ← getCloudFile(src_container  src_filename )
12:         dest_cloud_file ← getCloudFile(dest_container  dest_filename )
13:         FileCounter ← FileCounter + 1
14:         if src_cloud_file.hash = dest_cloud_file.hash then
15:             ComparisonCounter ← ComparisonCounter + 1
16:         end if
17:     end for
18:     if FileCounter = ComparisonCounter then
19:         return True
20:     end if
21:     return False
22: end procedure
```

function to identify a specific file in the Cloud. The algorithm then compares the hash value of both files and increments the *ComparisonCounter*. If all the files in both workflows are the same, the *ComparisonCounter* should be equal to the *FileCounter*, which counts the number of files produced by a workflow. Thus, it confirms that the workflows are repeated successfully. Otherwise, the algorithm returns false if both these counters are not equal. This means that the two given workflows have not produced the same outputs.

4.3 Reproduce Workflow Execution using ReCAP

In Section 4.1, the job-to-Cloud resource mapping using provenance information has been discussed. This mapping information is stored in the database for workflow reproducibility purposes. In order to reproduce a workflow execution, the researcher first needs to provide the wfID (workflow ID), which is assigned to every workflow in Pegasus, to the WF-Repeat component of the proposed framework to re-execute the given workflow. The WF-Repeat component retrieves the given workflow from the ReCAP Store (step 2(a) in Figure 4.7) along with the Cloud resource mapping stored against this workflow (step 2(b) in Figure 4.7). Using this mapping information, it retrieves the resource flavour and image configurations, and provisions the resources (step 3 in Figure 4.7) on the Cloud using the CRM component. Once resources are provisioned, it submits the workflow (step 5) using the underlying workflow management system.

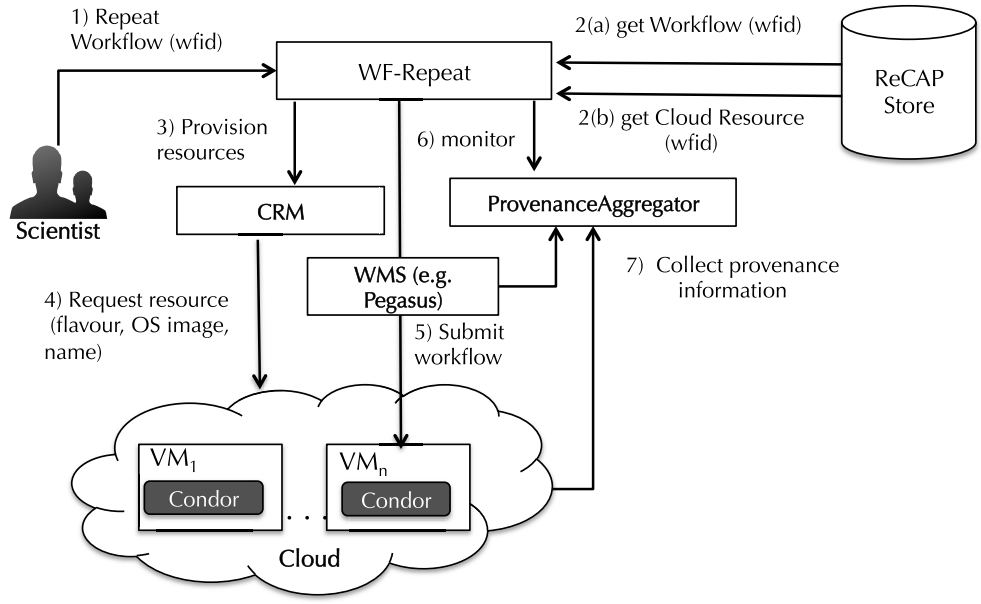


Figure 4.7: Illustrating the flow of activities while reproducing a workflow execution on Cloud

At this stage, a new workflow ID is assigned to this newly submitted workflow. This new wfID is passed over to the ProvenanceAggregator component to monitor (step 6) the execution of the workflow and start collecting its Cloud-aware provenance information (see line in Algorithm 9). It

is important to re-collect the provenance of the repeated workflow, as this will enable us to verify the provisioned resources by comparing their resource configurations with the old resource configuration. The Algorithm 9 shows the working of this approach.

Algorithm 9 Repeating a Workflow Execution on the Cloud using Cloud-aware provenance

Require: $wfID$: ID assigned to a workflow.

```
1:  procedure RepeatWorkflow(wfID)
2:      workflow ← getWorkflow(wfID)
3:      cloudResources ← getCloudResourcesForWF(wfID)
4:      for all resource ∈ cloudResources do
5:          name ← resource.name
6:          flavor ← resource.flavor
7:          image ← resource.image
8:          CRM.provisionResource(name, flavour, osimage)
9:      end for
10:     new_wfid ← submitWorkflow(workflow)
11:     ProvenanceAggregator.monitor(new_wfid)  ▷ Monitor the newly submit workflow
12: end procedure
```

4.4 Cloud-based Workflow Manager

In order to perform dynamic resource provisioning for workflow execution, there are two possibilities that could have been considered. One possibility is to provision all the resources on Cloud at once at the beginning of the workflow execution. This would create the similar number of virtual machines on the Cloud for a workflow re-execution that were used in its original execution. This approach is very much in line with the current execution scenario with Pegasus and other workflow management systems, which rely upon a static environment. This approach will be used in experiments using Pegasus to reproduce a workflow execution using the CAP information. However, in a dynamic scenario in which a resource is provisioned on-demand (when required) is to be used, a different mechanism is required that facilitates this. Such a mechanism is not available in Pegasus at the moment. In order to achieve this, a Cloud-based Workflow Manager (CWManager) has been developed that treats a given workflow as a graph and traverses it to determine the jobs ready for submission. This gives the flexibility to inject resource provisioning on the Cloud before the ready jobs are sent for submission. As CWManager is very basic in its features set, no advance workflow planning i.e. task clustering, task cleansing etc., is performed and no intermediary data management tasks are added to the workflow except for the two data management tasks used in the start of workflow execution. The overall design and internal working of the workflow manager is illustrated in Figure 4.8.

For performing one aspect of this research study i.e. the dynamic Cloud scenario, this workflow

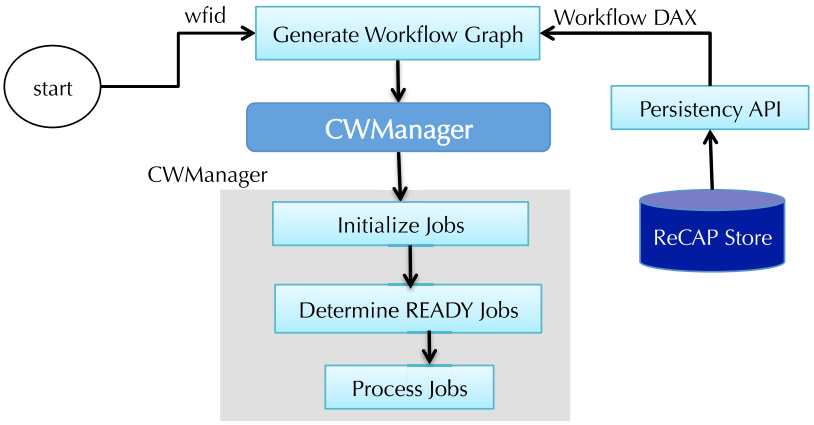


Figure 4.8: A simple Cloud-based Workflow Manager

manager interacts with the ReCAP database and retrieves the workflow description in DAX format for the given wf ID. A user can also provide the workflow DAX directly to it. From this description, it constructs a graph of the given workflow with the similar inter-job dependencies. The nodes of the graph represent the workflow jobs and the edges represent the inter-job dependencies. In this stage, it also adds two initial data management tasks which are designed to prepare the data directories on the storage location that will be used later on by the workflow jobs for reading or writing data. Before processing the workflow, all the jobs in the graph are set to an initial state i.e. WAITING. The jobs then go into the READY state i.e. a job is ready to be submitted. This state is determined with one rule i.e. "a job is ready to be submitted (when its parents have successfully completed)." (Chen and Deelman 2011, p. 2). To determine the jobs which are ready for submission and execution, the workflow manager retrieves the list of WAITING jobs and iterates over them. Only those jobs whose parent jobs are marked as COMPLETED are selected as READY jobs.

After identifying the ready jobs, the workflow manager starts processing them. Based on the job specification provided in DAX, the workflow manager creates a job submission script. This script contains the information about the input file, output files and actual job executable. This script also provides commands that assist in data management operations i.e. downloading the inputs from and uploading the outputs to a shared filesystem such as Network File System (NFS)[3] or a Cloud storage service such as OpenStack's Swift[4]. For testing, the Cloud storage service has been used to upload and download data. This job script can then be transferred to the provisioned Cloud resource i.e. the virtual machine and its execution starts at that point.

After this, the workflow manager initiates a resource provisioning mechanism through Resource-Provisioner to launch a new resource on which this job can be executed. Two different resource provisioning schemes i.e. (a) random and (b) fixed are supported. In the random scheme, resource configuration i.e. Flavour is selected randomly for the new resource. In the fixed scheme, a con-

[3]https://www.freebsd.org/doc/handbook/network-nfs.html
[4]http://swift.openstack.org/

figured i.e. fixed Flavour is used to provision every new resource. This process is illustrated in Figure 4.9.

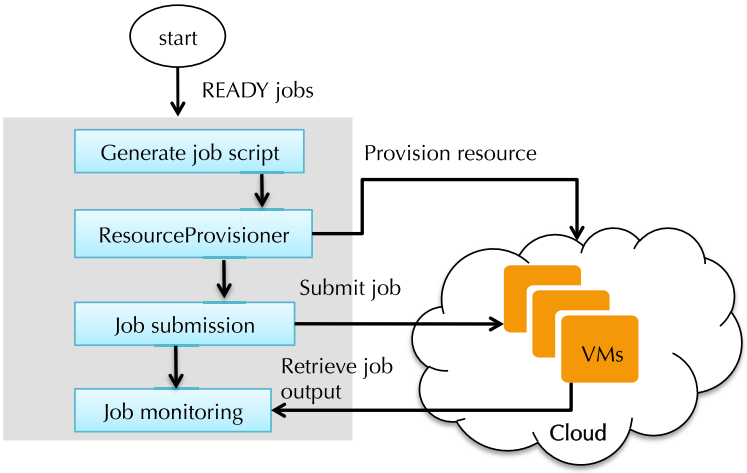


Figure 4.9: Processing READY jobs in CWManager

Once a job has been submitted, it is monitored to retrieve job output and virtual machine information. The monitoring process retrieves the job's stdout and stderr along with the virtual machine information and stores them in a log file. This log file then serves as the provenance data for this particular job. The collected provenance also includes Cloud resource configuration information, job start time, finish time, virtual resource boot up time and job status.

4.5 Integration with CRISTAL

In N4U, a user submits workflows on the N4U Grid infrastructure using CRISTAL (as discussed earlier in Section 3.5.2). The workflow submission is achieved through a Pipeline Service that was developed for the N4U Grid infrastructure. At present, the retrieved provenance information from the Pipeline Service does not provide any detail about the machine on which a job was executed. Collecting such information becomes challenging as there is no direct interface available in N4U or its developed services and the underlying infrastructure is Grid-based. In order to provide this support and collect machine information, a pegasus-kickstart[5] like approach has been designed by modifying the GridBroker component.

The GridBroker is responsible for constructing a workflow specification compatible with the Pipeline Service. While generating the workflow, it attaches a launcher script as a job's executable that launches the actual scripts (i.e. the actual job) after downloading them from the N4U infrastructure. The devised mechanism adds a monitoring Python code **(MonitorProcess.py)**, which collects the machine and process information, as a part of the job payload. Since the code is written using

[5]pegasus-kickstart: http://pegasus.isi.edu/mapper/docs/4.2/cli-pegasus-kickstart.php

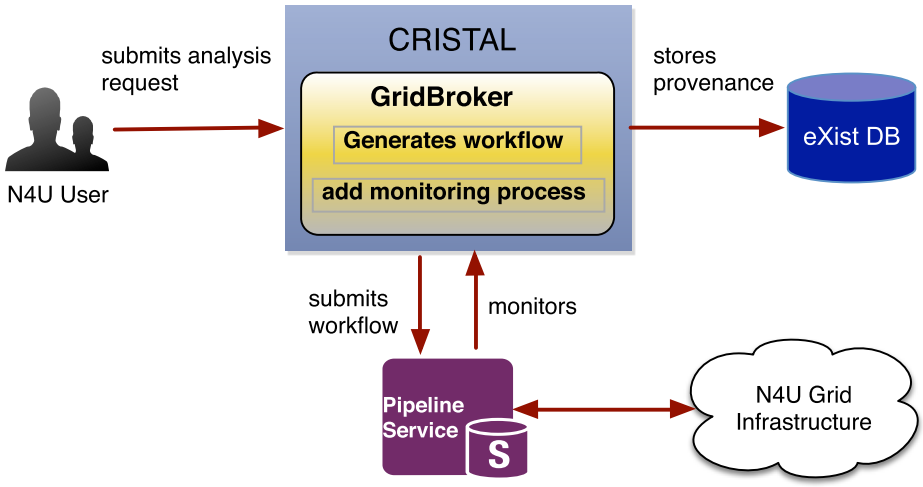


Figure 4.10: GridBroker adds a monitoring process in the job payload of a user analysis

Python's core libraries, and almost all Linux distributions come with a pre-installed Python environment[6], it is safe to assume that the code will not break during execution. The MonitorProcess.py looks for the CPU, CPU specs (speed, architecture, vendor), RAM, installed OS and its architecture and kernel version etc. and generates an XML file, as shown in Appendix B.1. This file is retrieved as part of the job output and stored in the CRISTAL database. Although, one cannot determine the VM image used to provision this resource as is the case in Cloud, however this data can give some information about the operating system i.e. Linux kernel running on the machine. For reproducing the workflow execution, the workflow scheduling or mapping algorithm can look up the collected provenance information for each job and ask for the similar resource from the Grid. The updated design is illustrated in Figure 4.10.

To fully retrieve the mapping, the SNoHi approach (discussed in Section 4.1.4) can be used in this case. Since N4U does not provide access to its Cloud IaaS services due to the security and policy reasons, the testing is not possible. However, the retrieved monitoring XML (as shown in Appendix B.1) contains information about the machine IP. Once this information is available, other mapping approaches such as Static or Lazy can establish the required mapping.

4.6 Summary

Based on the ReCAP architecture presented in Chapter 3, this chapter discussed its implementation details. Before discussing the mapping approaches to capture the Cloud-aware provenance, different Cloud usage scenarios were identified and discussed in Section 4.1. Based on these Cloud scenarios, different mapping algorithms have been discussed. The Static mapping algorithm has been devised

[6]https://docs.Python.org/2/using/unix.html

to establish job-to-Cloud resource mapping in existing Cloud environment, which is based on virtual cluster. However, this approach will not work in more dynamic Cloud environments in which resources are provisioned for a task and then destroyed when the task is finished. For this, another mapping algorithm, *Eager*, has been proposed. This approach attempts to establish job-to-Cloud resource mapping as early as possible. In doing so, it relies upon information coming from Condor. This dependence is due to Pegasus since it uses Condor for job execution. To avoid this dependency, the Lazy approach has been proposed, which registers the current running virtual machines' metadata in a temporary store and establishes the job-to-Cloud resource mapping later when the workflow execution completes.

All aforementioned mapping approaches rely upon a hostname or host IP information being available from the workflow management system. However, they will not work for systems such as Chimera which do not register this information. To tackle such cases, another mapping approach has been devised. This mapping approach makes use of augmented job scripts to log machine information. It then parses the logs to retrieve the job's host information, which is then linked to the registered VM information to establish job-to-resource mapping. These presented mapping approaches help in answering the **Research Question 2** (discussed in Section 1.4). After this, the provenance graph comparison approaches (structure, workflow output, virtual resources) etc. have been discussed.

Since the Pegasus does not dynamically provisioned resources on the Cloud, a Cloud-based Workflow Manager (CWM) has been devised and discussed. This enables the dynamic provisioning of resources with the given resource configurations for the jobs ready for execution. It can traverse the workflow DAG and provision the resources when required. The jobs are submitted to the newly provisioned resources. Once the jobs are finished, the Cloud resources are destroyed and the collected provenance is logged into a file.

The next chapter will present the evaluation strategy and experiments, which will be conducted to verify the prototype. Various aspects of the prototype such as mapping algorithms, workflow execution on different Cloud configurations, provenance comparison and provenance overhead will be evaluated.

Chapter 5

Evaluation and Experimental Environment

This chapter presents an evaluation methodology that was designed to evaluate the proposed system. It discusses a set of tests and the experimental environment used to perform the tests. This evaluation methodology is designed to validate the overall proposed approach and its key components such as the mapping algorithms, workflow reproducibility and workflow provenance comparison. It then discusses the identified experiments and expected outputs that will be discussed and analysed in Chapter 6. It also provides detailed information about the test environment and the framework used to perform the experiments in order to validate the work carried out in this thesis. The structure of this chapter is as follow. Section 5.1 revisits the hypothesis and research questions that this work is based upon. This provides an overview of what has already been accomplished in this thesis. Following this, the evaluation methodology is presented that identifies different types of tests required to validate this work. This chapter then discusses the experimental environment and experiments used to perform these tests.

5.1 Evaluation Process

The aim of this research thesis has been to answer the research hypothesis and to resolve the research questions that were presented in Chapter 1. The research questions that have driven this work are:

1. To what extent are the existing provenance capturing techniques adequate for capturing provenance of a Cloud-based scientific workflow?
2. How can provenance information be captured at different layers in a Cloud environment and interlinked?
3. How can correctness and completeness of the collected provenance be measured?

4. How can we demonstrate workflow reproducibility on the Cloud infrastructure using Cloud-aware provenance?

Research question number one was considered in Chapter 1 where existing literature in the domain of workflows, provenance systems in Grid and Cloud were analysed. The literature was analysed in the light of the main aim of this research study which focused on workflow execution on the Cloud. It was found that the existing work lacked the information about the configuration of the Cloud resources which were used for workflow execution. Since the Cloud is dynamic and resources are provisioned on-demand by requesting them from the Cloud middleware, the absence of Cloud resource information would make provisioning same execution infrastructure difficult. In this regard, the essential Cloud parameters to achieve workflow reproducibility on the Cloud were identified and discussed in Chapter 3. Chapter 3 also discussed the workflow reproducibility requirements for the Cloud environment. These requirements were identified and collected after reviewing the existing literature in this domain.

Research question number two was partially answered in Chapter 2 where provenance techniques in Cloud environment were analysed. It was found that different systems and approaches were designed that could capture information from different layers such as the hypervisor level, file system level and also workflow level. However, these approaches did not combine workflow provenance with the Cloud infrastructure information. In order to capture the required Cloud infrastructure information and to interlink it with the workflow provenance, multiple mapping approaches have been proposed and discussed in Chapter 4. Research question three was considered in Chapter 3 where correctness and completeness of provenance information were defined by evaluating the literature. These definitions were revised by keeping the objective of this research work in perspective. The correctness and completeness of the Cloud-aware provenance was then considered in the proposed solution by devising two algorithms (discussed in Section 3.8.7).

In order to address the final research question, a thorough evaluation of the proposed framework, ReCAP, is required. In this evaluation, different components and theoretical concepts proposed in ReCAP are tested through different experiments. These experiments verify the impact of different Cloud resource parameters on a job and also on a workflow execution performance. These experiments also evaluate the working of proposed mapping approaches such as the Static and Eager mapping (previously discussed in Section 4.1). Through these tests, the need to an Eager mapping approach has also been highlighted. By analysing the results of these experiments, the functioning of ReCAP and its ability to reproduce a workflow execution has been discussed in Chapter 6. Following section discusses different set of tests devised to evaluate different facets of ReCAP prototype which eventually help in answering the research hypothesis.

5.1.1 Evaluation Methodology

In order to verify the proposed approach, ReCAP, it was important to verify that all its components and algorithms were performing according to the theoretical understanding of the design. Since this proposed approach collected Cloud-aware provenance, it was also important to verify that the captured Cloud resource configurations could indeed affect a job and thus were required in the Cloud environment. For the complete evaluation of the research work, following tests were considered.

The first set of tests (discussed in Section 5.3.1) dealt with the validation of the captured information related to the hardware and software configurations of the Cloud resources used to execute a workflow. The aim of these tests was to verify the significance of different Cloud resource configurations on the job success rate, job performance and also on the workflow performance. The results of these tests justified the argument made in Chapter 1 and 3 for collecting the Cloud infrastructure information. Section 5.3.1 discussed the experiments performed to evaluate these tests.

In this thesis, two Cloud environments namely Static and Dynamic, that can be used to execute a workflow on the Cloud, have been discussed. For these two environments different job-to-Cloud resource mapping approaches (previously discussed in Section 4.1) have been devised. It was important to evaluate the functioning of these approaches and also to verify the theoretical understanding behind the approaches devised for the Dynamic environment in the Cloud. An experimental setup was presented in Section 5.3.2 to verify the importance and performance of the proposed mapping algorithms.

The third type of tests dealt with the correctness and completeness of the capture Cloud-aware provenance that is also the Research question number 3. Since the proposed approach in ReCAP relies upon the Cloud-aware provenance information, it is necessary to confirm that the capture information is correct and complete. The provenance correctness test evaluated the presence of all Cloud parameters, which were identified in Section 3.3), for a job. Any missing Cloud parameter in the provenance meant that the captured provenance was not correct. The provenance completeness was evaluated to verify that ReCAP could capture and establish job-to-Cloud resource mapping for all the jobs executed in a workflow. In case of missing job or missing Cloud provenance for a job, the completeness test would fail. The purpose of these tests was to verify that ReCAP could collect the Cloud infrastructure information. In Section 5.3.3, different scenarios to evaluate the working of provenance correctness and completeness algorithms (previously discussed in Section 3.8.7).

The fourth type of test (discussed in Section 5.3.4) was performed to reproduce a workflow using Cloud-aware provenance. This test verified the ability of the proposed approach in re-provisioning the execution infrastructure on the Cloud, used in earlier workflow execution, using the Cloud-aware provenance as captured by the proposed approach. The outcome of this test was verified by comparing the captured Cloud-aware provenance of both workflow executions i.e., the original

and the reproduced executions. The collected provenance of both executions should show the same resource configurations used on the Cloud.

The fifth set of experiments (discussed in Section 5.3.5) was designed to verify the provenance comparison approach discussed in Section 3.3. The captured provenance graphs were compared at different levels; at the structural comparison, output comparison and infrastructure comparison levels. The results of these comparisons enabled a researcher to confirm the reproducibility of a workflow execution on the Cloud. Moreover, these experiments also evaluated the working of the devised comparison algorithms (discussed in Section 4.2 and 4.2.3).

The final set of experiments dealt with the overheads (both performance and size) caused by the proposed mapping approaches. The purpose of these experiments was to verify the impact of the devised mapping approaches on the workflow execution. These tests also measured the size on disk required to store the Cloud-aware provenance information.

5.2 Experimental Environment

In order to evaluate the aforementioned tests, a Python based prototype was developed for ReCAP. Since Python is an interpreted language, it helps in fast development and testing of the prototype. Moreover, Python is supported on all major platforms which makes the code more portable. Since ReCAP needs interaction with the Cloud middleware to monitor and capture Cloud resource information, Apache Libcloud API was used. The Apache Libcloud API provides a middleware agnostic API which means it can interact with multiple Cloud providers. The Cloud infrastructure used during the evaluation of this research work was based on OpenStack middleware. The details of the execution environment is discussed in following Section 5.2.1. The aim of this research study is to reproduce workflow execution, so different workflows have also been identified. These workflows are used to evaluate the hypothesis of this research. The purpose of using different type of workflows is to showcase the capability of ReCAP in handling different types of workflows with different structures and sizes. Out of these workflows, the ReconAll workflow has been used in the N4U project. The details of these workflows are discussed in Section 5.2.2.

5.2.1 Execution Infrastructure

In order to carry out the experiments and workflow execution on Cloud, a Cloud-based execution infrastructure was used in this research study. Pegasus has been used as a workflow management system to submit and monitor workflow execution. The workflow execution took place on a virtual cluster, composed of virtual machines running Condor instances. This means that Pegasus had access to a Condor pool established over a set of virtual machines. These virtual machines were hosted on

a Cloud infrastructure offered by the Open Science Data Cloud (OSDC). This Cloud infrastructure uses OpenStack to offer IaaS services. However, OSDC did not provide a Cloud-based object storage service such as Swift provided by OpenStack.

To support data over a Cloud-based storage service, OpenStack's swift service along with other services was deployed locally at UWE servers. Since all services of OpenStack supports RESTful interfaces, this service can also be called from RESTful clients. As previously mentioned, the Re-CAP prototype uses an Apache Libcloud API to interact with the Cloud middleware. This API supports multiple Cloud middlewares and Openstack is one of them. Using this API, the Cloud Layer component can interact with OpenStack's compute and storage services.

5.2.2 Test Workflows

Before discussing the sample workflows, it is important to analyse the structure of different workflows used in various scientific communities such as neuro-imaging (in N4U) and astronomy. Two workflow examples i.e. (a) Montage workflow and (b) ReconAll workflow from astronomy and the N4U project have been discussed in Sections 5.2.2.1 and 5.2.2.2 respectively. From these workflow examples, different structural patterns are derived. These patterns show the inter-job dependencies. These patterns can then aid in devising the structure for the simulated workflow, which is discussed in Section 5.2.2.3. It also discusses the nature of the workflow, number of jobs in each workflow and the data used in them.

5.2.2.1 Montage Workflow

This workflow uses the components of Montage[1], a widely mentioned astronomy application to build mosaics of the sky by stitching together multiple input images (Sakellariou, Zhao and Deelman 2010). Montage is a general engine for computing mosaics of input images. The input images for the mosaics are taken from image archives such as the Two Micron All Sky Survey (2MASS)[2] and Sloan Digital Sky Survey (SDSS)[3]. The input images are first re-projected to the coordinate space of the output mosaic, the re-projected images are then background rectified and finally co-added to create the final output mosaic. The tasks in the workflow are depicted by the vertices in the graph and the edges represent the data dependencies between the tasks in the workflow. The workflow shown in Figure 5.1 has 20 jobs. The size of a Montage workflow depends on the number of images used in constructing the desired mosaic of the sky. The structure of the workflow changes to accommodate increases in the number of inputs, which corresponds to an increase in the number of computational jobs (Juve et al. 2013). This workflow is deterministic meaning that it produces the same outputs

[1] http://montage.ipac.caltech.edu/
[2] http://www.ipac.caltech.edu/2mass
[3] http://www.sdss.org

given the same inputs (Groth et al. 2009). This workflow has been used in various research articles (e.g. Deelman et al. 2008a; Hoffa et al. 2008; Sakellariou, Zhao and Deelman 2010).

There are multiple jobs used in a Montage workflow as shown in Figure 5.1. The mProjectPP job re-projects the given input images and the number of mProjectPP jobs is equal to the number of input images. It generates re-projected image and an 'area' image that consists of the fraction of the image that belongs in the final mosaic. These output images are then processed together in subsequent steps. The mDiffFit jobs compute a difference for each pair of overlapping images. The number of mDiffFit jobs in the Montage workflow depends on how the input images overlap. The resultant difference images are then fitted using a lease square algorithm by the mConcatFit job (Juve et al. 2013). The mConcatFit job is a computationally intensive data integration job. This is followed by the mBgModel job, which computes a correction to be applied to each image to obtain a global fit. This background correction is applied to each individual image at the next level of the workflow. The mImgTbl job aggregates metadata from all the images and creates a table that may be used by other jobs in the workflow. The mAdd job combines all the re-projected images to generate the final mosaic in Flexible Image Transport System (FITS) format as well as an area image that may be used in further computation. The mAdd job is the most computational intensive job in this workflow. Finally, the mShrink job reduces the size of the previously produced FITS image by averaging blocks of pixels and the shrunken image is converted to JPEG format by the mJPEG job.

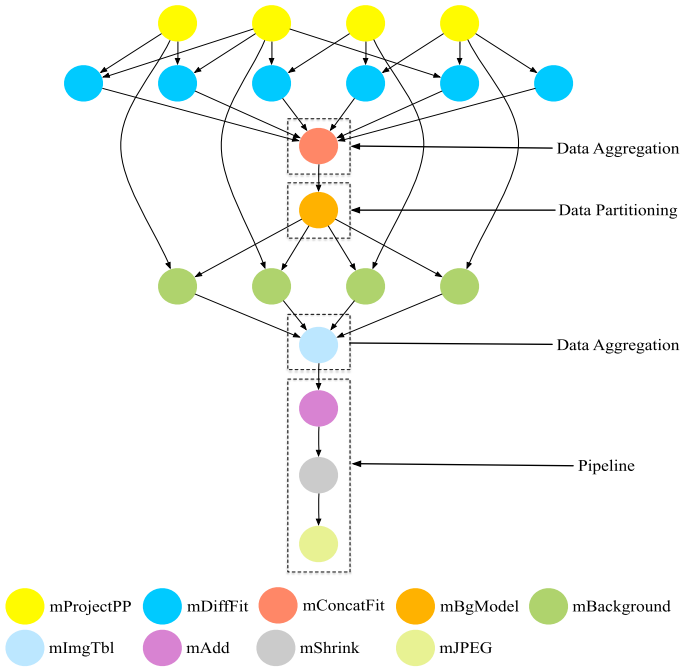


Figure 5.1: Illustrating the Montage workflow (original source: Pegasus workflows[4]. Used with permission of the Author)

[4]Pegasus workflow examples: https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

As it can be seen in Figure 5.1 that the Montage workflow has jobs such as mConcatFit and mBgModel that perform data aggregation (i.e. merge) or distribution (i.e. split). In a way, these jobs exhibit one-to-many jobs (i.e. split) relation and many-to-one job (i.e. merge) relations respectively. A similar pattern can be found in other workflows such as LIGO (Juve et al. 2013) and the CMS reconstruction workflow (Hasham et al. 2011). These two relationships among jobs have also been used in the simulated workflow (discussed in Section 5.2.2.3). Montage is a data-intensive application. The input images, the intermediate files produced during the execution of the workflow and the output mosaic are of considerable size and require significant storage resources (Deelman et al. 2008a). The Montage workflow used in the experiment contains 35 jobs and required eight input image files. This workflow produces 4 output files including one mosaic image file in JPEG format.

5.2.2.2 ReconAll Workflow used in N4U

This workflow, or also known as pipeline, is used in N4U project to reconstruct the given MRI scan image of a subject. This workflow consists of only one executable i.e. *recon-all*[5]. In N4U, each workflow (or also called pipeline) is composed of one executable script. The executable script performs all the required operations on the given input files and produces the required output. For instance, recon-all[6], a Freesurfer tool, is used in a pipeline named as nG+FreeSurfer+5.3.0+ReconAll+v01.script. Freesurfer[7] is an open source suite for processing and analysing human brain MRI images. It provides a set of tools for analysis and visualization of structural and functional brain imaging data. It comprises a popular and freely available set of tools for deriving neuroanatomical volume and cortical thickness measurements from automated brain segmentation (Gronenschild et al. 2012). The $recon-all$ pipeline takes one brain MRI image as input file in .nii format and performs the reconstruction process over it using the recon-all command available in Freesurfer. It is a batch program that performs all i.e. >30 steps such as Motion Correction and Cortical Parcellation Statistics, or any part of the FreeSurfer cortical reconstruction process. Depending upon the arguments supplied to the recon-all, various steps are executed accordingly. In order to perform all the steps on the given input neuroimage, *recon-all -all* is used. In N4U, one input file is provided to this script for analysis. Figure 5.2 depicts the graphical representation of this workflow.

5.2.2.3 Simulated workflows

In order to generate a simulated workflow that could exhibit the structural inter-job dependencies, it is important to analyse some of the existing workflows that have been used in the N4U community and also in other domains such as Montage used in astronomy.

[5]https://surfer.nmr.mgh.harvard.edu/fswiki/recon-all
[6]https://surfer.nmr.mgh.harvard.edu/fswiki/recon-all
[7]http://freesurfer.net/

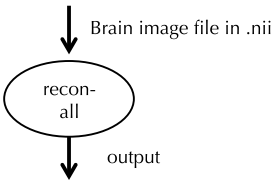


Figure 5.2: A graphical illustration of the ReconAll workflow used in N4U

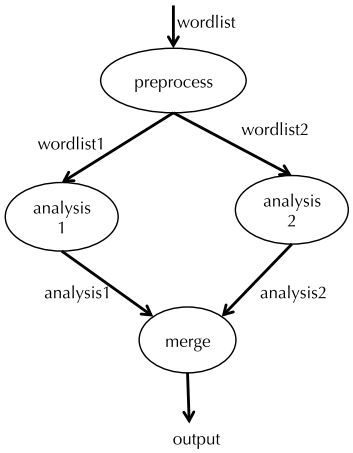


Figure 5.3: A test Wordcount workflow showing both split and merge characteristics of a workflow

The workflow used in N4U is very simple as it only has one job. On the other hand, the Montage workflow (discussed in Section 5.2.2.1) is more complex. It has split as well as merge jobs. Based on this understanding, a basic Wordcount workflow application composed of four jobs was written (shown below). This workflow had both control and data dependencies (Ramakrishnan and Plale 2010) among its jobs, which is a common characteristic in scientific workflows such as Montage or LIGO Inspiral Analysis (Juve et al. 2013). It also has split and merge job relations that have been shown in the Montage workflow. The first job (the Split job) took a text file and split it into two files of almost equal length. Later, two jobs (the Analysis jobs) were applied; each of these takes one file as input, and then calculates the number of words in the given file. The fourth job (the Merge job) took the outputs of earlier analysis jobs and calculated the final result i.e. total number of words in both files.

5.2.3 Prepared OS Images

As discussed in earlier chapters, the software stack used in a virtual machine is defined by the image used to provision that resource. Since this research work relies on OS image to provide the required software for job execution, a few OS images were prepared. In order to execute the ReconAll workflow, an OS image was required that could run the *recon-all* executable. To achieve this objective, the FreeSurfer (version 5.3) software was installed on an Ubuntu 12.04 operating system. Similarly, the Montage software was installed on a machine to prepare an OS image to execute the Montage

workflow. The simulated workflows do not require special software except Python and the Python is supported on all major Linux distributions. Because the workflow execution was performed through Pegasus, all OS images also had Condor installed.

5.3 Experiments

This section and its subsections describe the experiments performed using the proposed approach in the Cloud infrastructure. These experiments have been designed in order to validate the arguments for Cloud resource configuration and also to validate different aspects of the proposed system, ReCAP. The results of these experiments have been analysed and discussed in Chapter 6.

5.3.1 Experiment 1: Resource Configuration impact on Job

There are a few identified parameters (discussed in Section 3.3), which are essential in acquiring a resource on the Cloud. These parameters can affect a job success rate and also job execution performance. The job success rate is verified by changing the RAM configuration in the Cloud resource. The effect of Cloud resources on the job execution performance is verified by using different MIPS configurations. Since this information is not yet provided by the Cloud providers, a simulation framework, WorkflowSim (Chen and Deelman 2012), has been used to test it. The effect of Cloud resource configuration on the workflow execution (discussed in Section 5.3.1.1) is performed by using different Cloud resource configuration for resource provisioning and then executing workflows on them. The outcome of these tests will aid in analysing and confirming the theoretical rationale for capturing these parameters on the Cloud.

To demonstrate the effect of RAM on job failure rate, an important factor in Cloud resource configuration, a basic memory-consuming job has been written in Python. The aim is to validate the presented argument in favour of collecting Cloud resource configuration as discussed in Section 3.3. Three resource configurations (a) m1.tiny, (b) m1.small and (c) m1.medium, each with 512 MB, 2048 MB and 4096 MB RAM respectively, have been used for this experiment. Table 5.1 shows the resource configurations.

Table 5.1: Cloud Flavours details

Flavour	vCPU	RAM	Hard Disk
m1.tiny	1	512 MB	0 GB
m1.small	1	2048 MB	20 GB
m1.medium	2	4096 MB	40 GB
m1.large	4	8192 MB	80 GB

To demonstrate the effect of CPU configuration of a Cloud resource on job performance, a basic compute-intensive job has also been written in Python. This job generates a Fibonacci number based

on the given input number. Three cloud resource configurations (a) m1.tiny, (b) m1.small and (c) m1.medium, shown in Table 5.1, are used for this experiment. To demonstrate the effect of multiple CPU cores, assigned to a virtual machine, on job performance, a basic parallelized compute-intensive job has been written.

To demonstrate the effect of CPU characteristics, most importantly MIPS, on job performance, a simulation has been performed using the WorkflowSim framework (Chen and Deelman 2012), which is written in Java. The purpose of this test is to evaluate the impact of CPU speed on the job performance and to ascertain whether this factor should be made part of the collected provenance. During the testing of the prototype, two different Condor clusters on two different Cloud infrastructures were configured. One Condor cluster was provisioned at UWE and the other Condor cluster was provisioned on OSDC. On analysing the clusters, it was found that both clusters offer different MIPS. Even in one cluster, different VMs hosted on same physical server exhibit different MIPS capability. Table 5.2 shows the MIPS for each machine in two Condor clusters. Due to this reason, this simulated experiment was performed to verify the impact of CPU speed or MIPS on the job execution performance. Following section describe the experiment prepared to evaluate the effect of the Cloud resource configuration on the workflow.

Table 5.2: Resource information of used Condor Pools

Cloud Provider	Arch	OS	RAM	MIPS	KFLOPS
UWE	$x86\_64$	Linux	2002	15369	1518351
	$x86\_64$	Linux	2002	15362	1494906
OSDC	$x86\_64$	Linux	2003	12583	1129282
	$x86\_64$	Linux	2003	12487	1146380
	$x86\_64$	Linux	2003	10938	1515023

5.3.1.1 Resource Configuration impact on Workflow

To demonstrate the overall effect on the workflow execution performance, sample workflows have been executed on randomly provisioned cloud resources and on resources using the CAP information. For this, the designed workflow manager i.e. CWManager is used (discussed in Section 4.4). During these tests, the sample workflow will be executed on the Cloud resources with different configuration settings. Three different types of configuration, (a) Random, (b) Tiny and (c) Small, are used to investigate their effect on workflow execution. In the Random configuration, all virtual machines have been provisioned using a randomly selected flavour. A flavour describes a resource configuration by providing information such as vCPU, RAM, Hard Disk etc. about a resource. In

the Tiny configuration, all virtual resources have been provisioned with m1.tiny flavour. In the Small configuration, all virtual machines have been provisioned with m1.small flavour. All these flavours are shown in Table 5.1.

These configurations have been selected to mimic different user behaviour on the Cloud for workflow execution. The Random configuration is selected to mimic the resource provisioning behaviour on the Cloud in which a user does not know the configurations in advance and thus provisions resources arbitrarily. The other two configurations i.e. Tiny and Small are used to mimic the user behaviour in which a user provisions all resources with a single configuration for workflow execution. The reason for selecting two configurations for this behaviour is to demonstrate the effect of different but fixed resource selection on workflow execution. All these resource provisioning behaviours for workflow execution on Cloud are different from the objective of this research study, which aims to take an informed decision about resource provisioning using the collected Cloud-aware provenance information.

5.3.2 Experiment 2: Job-to-Cloud Resource Mapping Approaches

The purpose of these experiments is to provide evidence for the identified two scenarios on the Cloud (discussed in Section 4.1) and also to demonstrate the working of the proposed mapping algorithms. Along with establishing the job-to-Cloud resource mapping, these experiments should also determine the time it takes Pegasus to register the VM information in its provenance database for each job execution. For this purpose, a data collection framework, the Host Information Availability Framework, has been designed that will assist in determining the time delay in recording the host information in the Pegasus database. The result of this experiment will justify the need for the Eager mapping algorithm.

5.3.2.1 Host Information Availability Framework

In order to determine the time delay, the availability of host information in Pegasus database has been monitored. A workflow composed of a single job has been submitted through Pegasus. The workflow structure (in DAX) of the sample workflow is shown in Appendix A.3. It has been a reported concern that a job faces many delays during its life cycle and one possible delay may be caused by determining the exit status after a job has finished its computation (Chen and Deelman 2011) on a resource. This delay can also affect the timely availability of host information on which a job finished execution. This information is very crucial for establishing the mapping between the job and the resource used on the Cloud. If this information has arrived late, it is possible that the resource on which a job was executed no longer exists on the Cloud, thus no mapping between the job and its Cloud resource can be achieved. For determining the existence of such a delay in Pegasus, the following framework is developed.

A very simple workflow with only a 'dummy' job has been designed (see Appendix A.3) for the experiment. This workflow has been executed through Pegasus on the Cloud resources. The dummy job mimics the pseudo-job processing by sleeping for a given time, the value of which is passed to it as an argument. In this workflow, the event notification mechanism provided by Pegasus (see lines 18) has been used to invoke the monitoring process. Out of available events in Pegasus, the START event has been selected. The reason for selecting the START event was because this provided a good measure of how long it took Pegasus to register the job to the resource (host) mapping in its database since the job had started its execution. This information was also helpful as the Eager approach attempted to acquire host information as early as possible due to the dynamic scenario of the Cloud (discussed in Section 4.1.2. The monitoring process began as soon as the dummy job started execution on a resource.

As a job was submitted for execution, Pegasus invoked the JobStartEventHandler (see Figure 5.4), a Python script, with job information. This triggered the host information monitoring in the Pegasus database for the finished job and workflow. The JobStartEventHandler registered the job information into a SQLite database[8] with the status 'new' for monitoring purposes. The monitoring was performed by another process which continuously looked for jobs to monitor with the status 'new'.

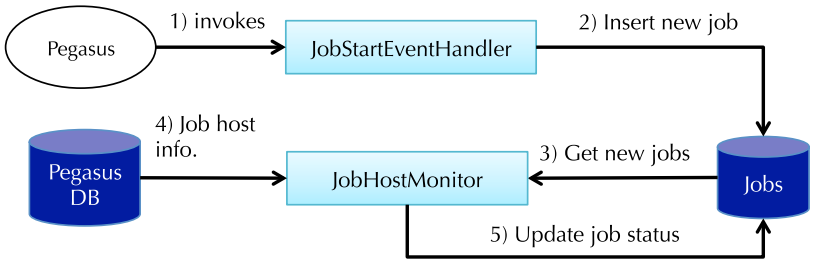


Figure 5.4: Job to Host monitoring framework

The monitoring program, JobHostMonitor, received the notification of a new job from the database and launched a new thread that started periodic monitoring of the job's host information in the Pegasus database. It also updated the status of this job from 'new' to 'processing' to avoid monitoring the same job again. The monitoring thread's polling interval was set to 2 seconds to avoid exhausting the CPU with 100% consumption. This process kept on monitoring the host information for a job until it was available. Upon finding the host information, the monitoring thread stopped and updated the job status in the SQLite database to 'Done', which means that monitoring for this particular job had been finished. The design of this test is shown in Figure 5.4.

[8]https://www.sqlite.org/

5.3.3 Experiment 3: Provenance Correctness and Completeness

The experiment for verifying the correctness and completeness of the captured provenance relied upon the job-to-Cloud resource mapping. In order to evaluate the working of the proposed approaches, different use cases related to stored mapping information have been envisaged. These use cases related to provenance correctness analysis are given below.

1. **Use Case 1**: The correctness check should return $True$ if all the Cloud parameters exist in the database for a given job.
2. **Use Case 2**: The correctness check should return $False$ if any of the Cloud parameters is missing.
3. **Use Case 3**: The correctness check should return $False$ if there are no Cloud parameters existing in the job-to-Cloud resource map.

From these use cases, only those Cloud-aware provenance records in the database were determined as correct for which all the Cloud parameters were present. In the absence of any or all Cloud parameters, that provenance was considered as incorrect. According to these use cases, existing job-to-Cloud resource mappings were deliberately modified to see their effect on the proposed algorithm. Its result has been discussed in Section 6.4.1. The correctness analysis worked on individual job level. However, the completeness analysis worked at the workflow level. It looked for the provenance information for all the jobs described in the workflow description and evaluated captured provenance. For this, another set of use cases have been designed.

1. **Use Case 4**: All the jobs described in the workflow description are present in the workflow provenance and all the jobs have correct Cloud-aware provenance. In this case, it should return $True$.
2. **Use Case 5**: The completeness analysis should return $False$ if there is a missing job in the workflow provenance, or if there is incorrect Cloud-aware provenance information.

5.3.4 Experiment 4: Workflow Reproducibility using ReCAP

To evaluate the reproducibility of a workflow, three different factors have been compared between the original workflow and its repeated workflow. These three factors are: (a) workflow structure, (b) workflow inputs and outputs and (c) the resources on which workflow jobs executed. By comparing all these three factors, we will be in a better position to say whether a workflow has been repeated successfully. To achieve this, provenance information combined with Cloud information was converted into a graph and then the provenance graph of two given workflows were compared, based on

the three factors mentioned above. It compared the nodes and edges of the graphs for a structural comparison. It then compared the inputs and outputs data using an Hashing algorithm to confirm whether the consumed or produced data were similar. It also compared the Cloud infrastructure on which workflow jobs were executed.

To test the workflow reproducibility approach of the proposed system, a Python based prototype was developed using Apache Libcloud[9] a library to interact with the Cloud middleware. To execute the workflows, a 20 core Cloud infrastructure was acquired from the OSDC [10] organisation. This Cloud infrastructure used the OpenStack middleware (openstack.org) to provide the infrastructure-as-a-Service capability. A small Condor cluster of three virtual machines was also configured. In this cluster, one machine was a master node, which was used to submit workflows, and the remaining two were compute nodes. These compute nodes were used to execute workflow jobs.

To evaluate the workflow reproducibility using the Cloud-aware Provenance (CAP), two sets of tests were conducted. In the first test, it is investigated whether the proposed system could capture and then re-provision the similar resources on the Cloud. For this experiment, the sample workflows were executed on the Condor cluster using Pegasus through the WMS WrapperService component (as discussed in Section 3.8.1). Once a workflow was finished and its job-to-resource mapping was available, a user could initiate the request to reproduce it. The WF-repeat component re-provisioned the resources on the Cloud using the collected CAP information and resubmitted the workflow using Pegasus, and started its monitoring. Since the aim of this study was to acquire a similar execution infrastructure for reproducibility, the experiment verified whether the collected Cloud-aware provenance for both workflow executions, original and reproduced, was the same. In this experiment, three workflows i.e. Wordcount, Montage and ReconAll were executed and re-executed using ReCAP.

For the second test, ReconAll workflow was re-executed but on different resource configuration in order to see its impact on the workflow execution performance. The captured Cloud-aware provenance is then compared with the earlier execution of the ReconAll workflow to determine the difference. This change in Cloud-aware provenance should also affect the infrastructure comparison and provenance structure comparison. This experiment will highlight the significance of Cloud-aware provenance.

As discussed in Section 3.3, the workflow output comparison has been identified as one of the mechanisms to determine workflow reproducibility. This experiment aimed to compare the outputs produced by two workflow executions (original and reproduced) using the Workflow Output Comparison algorithm (discussed in Section 4.2.3). In this experiment, the outputs produced by the sample workflows were compared against the output re-produced by the same workflows that were executed on re-provisioned resource using ReCAP.

[9] http://libcloud.apache.org
[10] https://www.opensciencedatacloud.org/

5.3.5 Experiment 5: Provenance Comparison

In order to verify the workflow provenance comparison mechanism, the following set of experiments was devised. These experiments validated the workflow provenance graph structure comparison algorithm (presented in Section 4.2), the workflow output comparison and also the Cloud infrastructure comparison. All these comparisons have been identified in the workflow reproducibility requirements (discussed in Section 3.3). These comparisons were performed to ensure different facets of workflow reproducibility.

In order to perform the workflow provenance graph structure comparison, different runs of the same workflows were compared against each other to detect variations. The devised Boolean algorithm (discussed in Section 4.2.1.2) should have been able to deduce that the given workflow provenance graphs were not structurally similar. The numerical comparison algorithm (discussed in Section 4.2.1.3), when applied, should have been able to provide a numerical value suggesting the degree of similarity between the given provenance graphs. The evaluation also carried out a performance evaluation of the algorithms by measuring the execution time on the increased size graphs. The increase in the graph size also increased the complexity of the given graph traversal.

In order to perform the workflow output comparison, different runs of a same workflow were provided to the comparison algorithm (discussed in Section 4.2.3). The algorithm should have been able to compare and identify similar outputs produced by the given workflows.

5.3.6 Experiment 6: Provenance Overhead

The purpose of the capturing provenance information was to facilitate a user with his intended work. However, it comes at the cost of overheads related to size (as discussed in Section 2.4.1) and performance. This set of experiments evaluated the impact of the proposed provenance capturing approach on the workflow execution time or individual job execution time, along with the data size requirements to store the Cloud-aware provenance information. In these experiments, sample workflows were executed with and without ReCAP mapping algorithms. In the absence of any ReCAP mapping algorithm, the job execution time was calculated and compared against the time when ReCAP mapping was enabled. This gives the performance overhead caused by the mapping algorithm to capture and interlink job to resource mapping information. Similarly, the data size required to capture job to resource mapping was also calculated. This helped in highlighting the additional space i.e., size overhead, required to store the Cloud-aware provenance using ReCAP.

As discussed in the mapping algorithms such as Eager and Lazy, and Static, the proposed approaches remained outside a virtual machine executing any workflow job. Since no ReCAP specific process was running inside a virtual machine, there should not have been any performance impact on the job caused by the provenance capturing mechanism. However, there could have been some

performance penalty caused by the SNoHi mapping approach (discussed in Section 4.1.4), which had been integrated and tested with CRISTAL, as it attempted to acquire machine information by running a Python code along with a job. The results of these experiments aid in analysing the overheads caused by the ReCAP mapping algorithms.

5.4 Summary

In this chapter, an evaluation process has been discussed that highlight the research questions and hypothesis. It then discussed an evaluation framework that presented a different set of experiments to verify the proposed approach. To evaluate the proposed approach, its constituent components were tested in order to verify that each proposed component was functioning according to the theoretical understanding presented in previous chapters. Since this research work focuses on workflow execution on the Cloud, a few tests were discussed that first verified the impact of the Cloud parameters on the job and also on the workflow execution. These tests provided the rationale in capturing Cloud-aware provenance in order to achieve workflow execution reproducibility. The impact of different resource configuration parameters on the workflow execution performance was also measured to highlight the significance of the collected Cloud resource configuration parameters. It then discussed the tests (presented in Section 5.3.2) for evaluating the need of different mapping approaches proposed in this thesis. The results (discussed later in Chapter 6) of these tests aided in justifying the mapping approaches designed for the Dynamic environment used in Cloud. Once the job-to-Cloud resource mapping was established using the proposed approaches, it was necessary to verify that the captured Cloud-aware provenance was correct and complete. These tests were important because without correct and complete provenance of a workflow, reprovisioning resource on Cloud with same configurations used in earlier execution was not possible. In this regard, a few tests were designed along with different use cases (discussed in Section 5.3.3) to verify the correctness and completeness of the captured provenance. The result of these tests aided in answering **Research Question 3**.

Since this research uses the Cloud-aware provenance in order to re-provision same execution infrastructure, this facet of the ReCAP was also evaluated. The Cloud-aware provenance of different workflows' executions were captured and compared with the Cloud-aware provenance captured for the reproduced workflows on the re-provisioned resources. From these tests, ReCAP's ability to re-provision similar execution resources on the Cloud was evaluated. The result of these tests aided in answering **Research Question 4**. Once a workflow was re-executed, a few more tests were conducted to perform the comparison of provenance graphs of original and reproduced workflow. In these tests (presented in Section 5.3.5), three aspects of the provenance comparison i.e. 1) structural similarity 2) infrastructure and 3) output, were evaluated. The results of these tests would provide a deep insight about the workflow execution and also aid in determining the functioning of the proposed comparison algorithms discussed in Chapter 4. By conducting all these tests on the workflows, their

combined result was used to determine the achieved workflow reproducibility. The thorough analysis of these results helped in resolving the **Research Question 4**.

Besides these tests, there were additional tests presented in Section 5.3.6 to measure the overhead caused by the Cloud-aware provenance. It was known from literature that provenance could affect in two ways i.e. size on disk and execution performance. The effect of capturing and mapping approaches on the execution was measured to determine the performance overhead. The size on disk overhead was also measured to determine the amount of space required to support the Cloud-aware provenance. By performing all these tests, which evaluated individual components of ReCAP and also overall workflow re-execution on the Cloud, a thorough evaluation of the proposed approach was performed. This not only helped in confirming the error free functioning of ReCAP and its components, but this would also aid in identifying area for improvements.

The next chapter, Chapter 6 presents and analyses the results obtained from the aforementioned experiments. The results discussion will explain the behaviour of the proposed system under different situations as discussed in this chapter.

Chapter 6

Results and Analysis

In this chapter, the results obtained from the designed experiments (presented in Chapter 5) will be discussed in order to evaluate the research hypothesis outlined in Chapter 1. These results will help us determine the functionality and performance of the proposed system under different test scenarios, as discussed in Section 5.3. This chapter first briefly re-iterate the experiments that were performed to evaluate the ReCAP prototype and research hypothesis. It is then followed by the analysis of the results obtained.

6.1 Evaluation Experiments

In order to evaluate the proposed approach, different set of experiments (previously discussed in Section 5.3) were designed. The purpose of these experiments was to verify the approaches devised in this research. As this research operates on Cloud-aware provenance, it is essential to verify that Cloud resource configurations can affect a job as well as workflow in various ways. The results of these experiments are discussed in Section 6.2.

ReCAP proposed multiple mapping approaches for different Cloud usage environments. These approaches can capture Cloud provenance and map it to the workflow provenance to generate Cloud-aware provenance. The next set of experiments was designed to verify the functioning of these approaches. They also aim to highlight the significance of the approaches such as *Eager* designed for the dynamic Cloud environment. The results of these experiments are discussed in Section 6.3.

Since ReCAP relies upon Cloud-aware provenance information for re-provisioning the resources to re-execute a workflow, it was important to verify that the collected provenance was correct and complete. In case of incomplete or incorrect provenance collected, ReCAP would not be able provision similar execution infrastructure for workflow re-execution. The analysis of these experiments is presented in Section 6.4. Once a complete provenance is captured, ReCAP can re-provision resource on the Cloud for workflow reproducibility. In order to verify this capability of ReCAP, the

test workflows were re-executed using ReCAP. Their provenance was collected and compared with the earlier executions to determine whether ReCAP had successfully provisioned the same resources on the Cloud. For this experiment, all three test workflows (discussed in Section 5.2.2) were executed using ReCAP.

As discussed in Section 3.3, the workflow reproducibility is determined by comparing the provenance graphs of both original and reproduced workflow. This comparison includes graph structural comparison, infrastructure comparison and output comparison. For these tests, the provenance traces of original workflows and their reproduced counterparts were provided to the comparison algorithms and their results are analysed in Section 6.6. Another purpose of this experiment was to verify that the comparison approaches perform according to the theoretical understanding that was presented in Chapter 4.

As previously discussed in Section 2.4.2 that provenance can cause size and performance overhead, it was important to see how much overhead ReCAP can cause. In this test, provenance information was collected using the proposed mapping approaches and their effect on the workflow execution was measured. Similarly, the disk space usage was also measured to determine the size overhead caused by ReCAP. The study of these results is presented in Section 6.8. The following sections present the results and their analysis.

6.2 Experiment 1: Resource Configuration impact on Job

This section discusses the impact of resource configuration on job performance and its failure rate. Various experiments were performed to analyse the effect of RAM on a job's failure, to analyse the effect of CPU on the job's performance and to analyse the effect of specific resource configurations on the workflow execution performance. This section also discusses the effect of including the CPU MIPS information in the Cloud-aware provenance.

6.2.1 Analyse the RAM effect

This experiment was designed to evaluate the significance of capturing the virtual machine's RAM parameter in the Cloud-aware provenance. As argued previously in Section 3.3 this parameter can affect a job's performance as well its failure rate; the following Figure 6.1 confirms the effect of the RAM parameter on the job's failure rate. Figure 6.1 shows that all jobs were successful on all resource configurations until the job's RAM requirement reaches 500 MB. As soon as the job's memory requirement approaches 500 MB, the job starts failing on the Cloud resource with the m1.tiny configuration because this resource configuration can only provide a maximum of 512 MB of RAM. This memory space is shared among the operating system processes and the job process,

consequently not enough memory is left for the job. On the other hand, the jobs executed on two other resource configurations i.e. those of m1.small and m1.medium, respectively offering 1024 MB and 2048 MB of RAM respectively, were all successful. In this experiment, each job was executed five times with the given memory requirement on each resource configuration. This specific experiment confirms that the RAM can play an important role in job's success rate. This factor is especially important for jobs processing large amounts of data and consequently require more RAM.

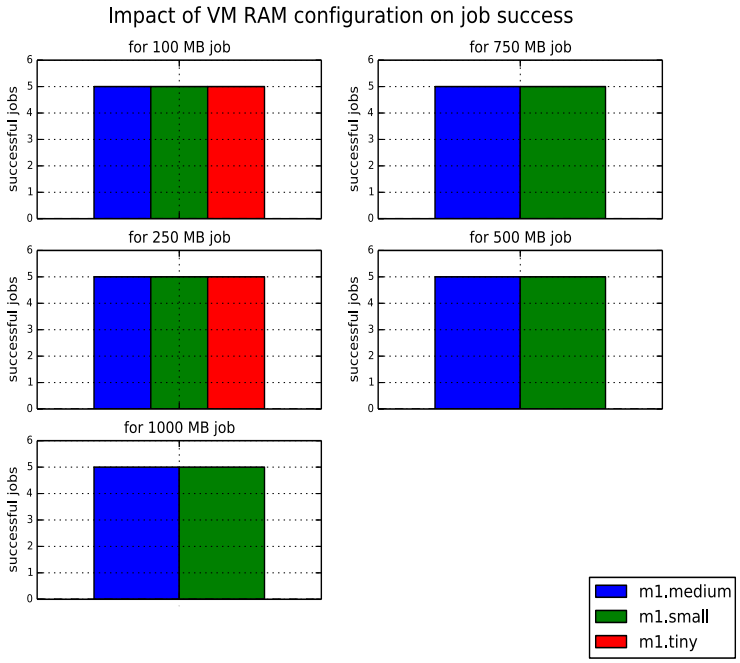


Figure 6.1: A Cloud resource's RAM configuration impact on job success

6.2.2 Analyse the CPU effect

In order to verify the CPU effect on the job performance, a compute intensive job calculating the Fibonacci number was written and executed on different resource configurations (shown in Table 6.1). The result shown in Figure 6.2 indicates that a job executed on the *m1.large* resource configuration performs better than the job executed on the other resource configurations. The m1.large resource configuration provides more CPU cores and more RAM to the job than the other two resource configurations i.e. m1.small and m1.medium. As can be seen in the figure, the impact of CPU is not evident for the lower ranges of Fibonacci number i.e. 20-30, 30-40. However, as the ranges increases, requiring more computation to calculate the Fibonacci number, the job executed on the improved resource configurations i.e. m1.medium and m1.large performed better. The job took less time on the *m1.large* resource for higher Fibonacci ranges i.e. 40-50 and 50-55.

Table 6.1: Resource Flavours used to execute the compute intensive job

Flavour	vCPU	RAM	Hard Disk
m1.small	1	1024 MB	10 GB
m1.medium	2	2048 MB	20 GB
m1.large	4	4096 MB	40 GB

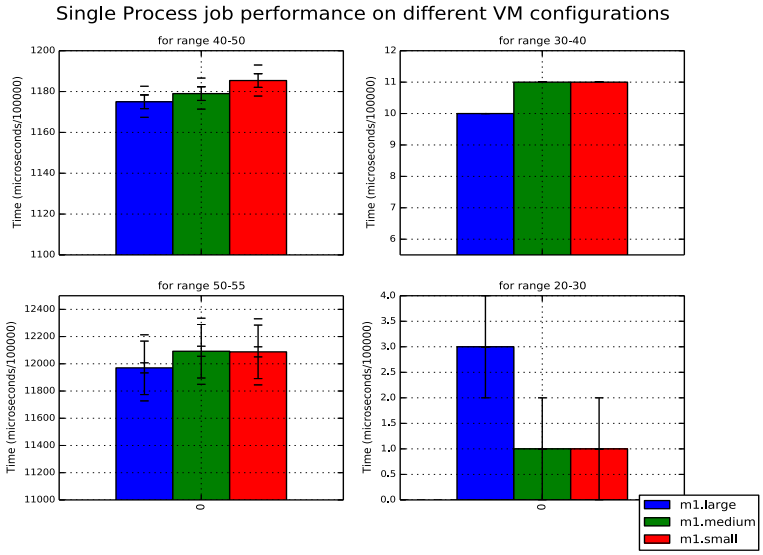


Figure 6.2: Single process job running on different resource configurations

Since Table 6.1 shows that a few resource configurations offer multiple CPU cores, another compute intensive job was written to calculate the Fibonacci number using parallel programming. This job parallelizes the computing on the available CPU cores on the virtual machine to calculate the Fibonacci number for a given range. The result in Figure 6.3 shows that a job running on improved resources with parallel programming performs better than a single process job on the same resource. Earlier Figure 6.2 provide an insight that the performance of a job improves on an improved resource with multiple CPUs. However, the performance of parallel job improves many folds on improved resources (as shown in Figure 6.3). This means that the number of CPUs in a resource can affect a job's performance that consequently can affect the overall workflow execution performance. This result also confirms that such information about the Cloud resource should be captured as part of the Cloud provenance. Moreover, this factor could play a key role for workflows e.g. the Epigenome workflow in the genomic experiments like (Ocana et al. 2011), which is both compute and data-intensive (Pietri et al. 2014), in which analysis performance is also important.

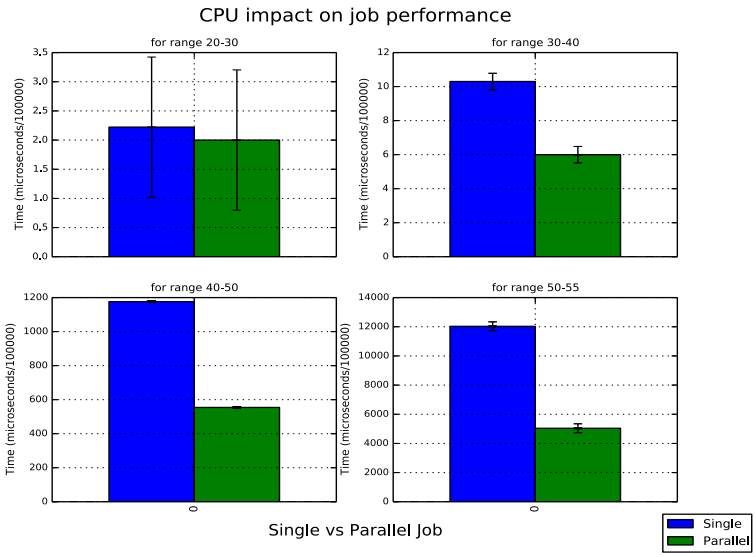


Figure 6.3: Single Process vs Multi-process job running on the *m1.large* resource configuration

6.2.3 CPU MIPS Impact on Workflow Execution performance

This experiment was performed in order to verify the impact of and to highlight the importance of collecting the CPU related data in gathering Cloud-aware provenance information. The motivation behind this experiment came from the findings during the workflow execution on the Condor cluster provisioned on the Cloud infrastructure. It was found that the virtual machines participating in the Condor cluster did not have the same MIPS value (calculated by the Condor benchmarking process) (see Table 5.2). Moreover, the existing Cloud APIs do not provide access to this information and the resource provisioning request to the Cloud infrastructure also does not include this information. Since this information is very low level and normally determined by benchmarking the resources (as is the case with Condor), this could be one reason that current APIs do not support it yet. On Amazon EC2 page[1] describing the available instance types (or flavours), one can find some indication of CPU speed. However, this information is not accessible through the API. Because of these reasons, this experiment was performed to help in building the support argument for including such information in collecting Cloud-aware provenance information.

Since existing Cloud APIs do not provide this information, a simulated workflow execution was conducted using the WorkflowSim framework. This framework allows a user to simulate a datacentre provisioned by describing the virtual resource in terms of the RAM, Hard Disk and CPU MIPS parameters. To mimic the behaviour of different MIPS values in virtual machines a randomly generated MIPS within a small range (12500 with variation of 1500), and a large range variation (10500 with variation of 4500) are assigned to the simulated VMs. Figures 6.4 and 6.5 show that execution time is directly affected by a change in the MIPS value. As the MIPS value increases, the

[1] https://aws.amazon.com/ec2/instance-types/

execution time decreases and vice versa. In this test, the same Wordcount workflow is simulated on the WorkflowSim but with the given hardcoded execution times.

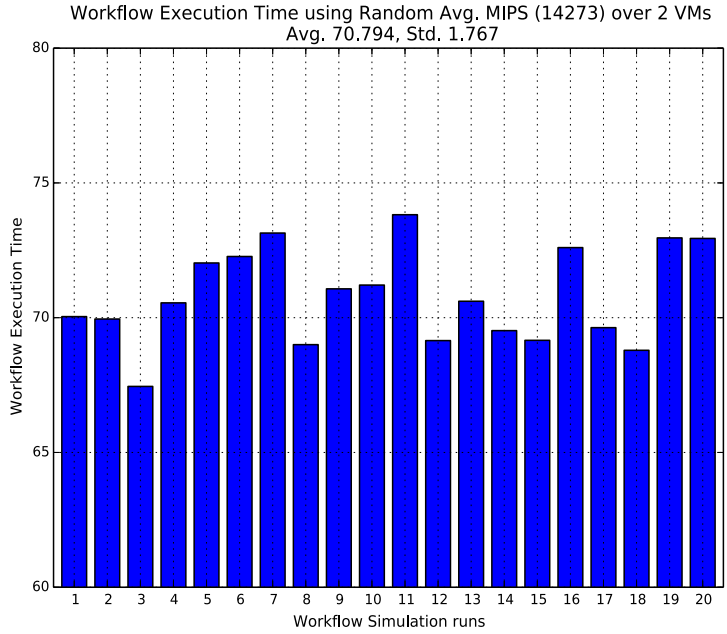


Figure 6.4: Effect of the CPU MIPS on the simulated workflow execution by randomly assigning MIPS from a range to the VMs

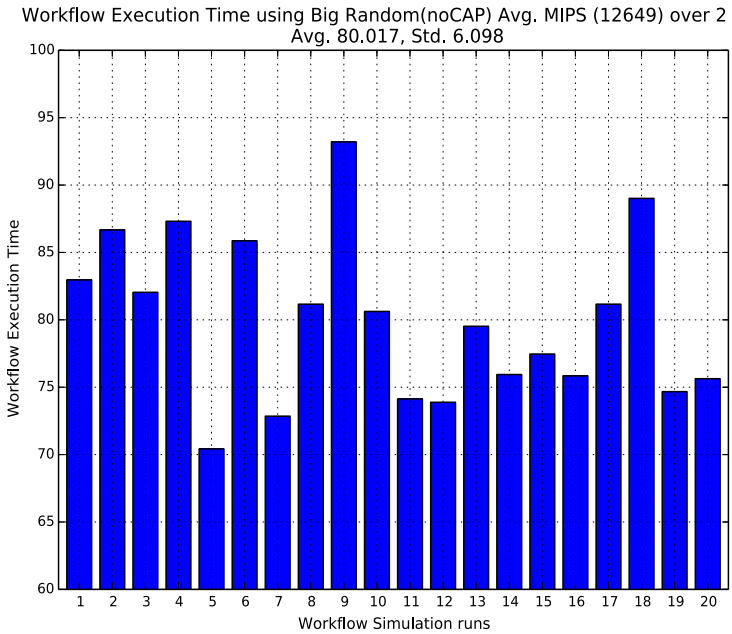


Figure 6.5: The effect of the decrease in the average CPU MIPS on the simulated workflow execution

As the MIPS value increases, the job execution time reduces which results in a better workflow execution performance as is shown in Figure 6.5. With this result, it can be concluded that the MIPS information along with the collected job-to-Cloud resource provenance is very important since it directly affects the job execution time. This information should also be provided while sharing or publishing the results as part of an experimental environment. Seeing the importance of MIPS, it can be argued and proposed that Cloud Providers should also look for ways to offer this as a configuration parameter to a user while provisioning a resource on the Cloud.

6.2.4 Cloud Resource Configuration impact on Workflow Performance

Previous experiments evaluated the impact of different resource configurations at the job level. The following set of results aim to show the impact of Cloud resource configuration at the workflow level. As discussed in Section 5.3.1.1, the sample workflow i.e. Wordcount was executed at least five times for each resource configuration. The workflow was executed using the workflow manager, discussed in Section 4.4, that is capable of traversing the workflow graphs and provisioning the Cloud resources on demand. Figure 6.6 shows the average workflow execution time for each configuration type and the small red lines in the figure represent the error bar calculated as standard deviation. Figure 6.6 shows that the average workflow execution time for the Tiny configuration (i.e. using the m1.tiny flavour type) is higher than the average workflow execution time for the Small and Random configurations. Since the virtual machines provisioned with the m1.tiny configuration have less RAM and fewer hard disk resources than the machines with the Small configuration, this is reflected on each job execution time and eventually on the overall workflow execution performance.

Figure 6.6: Average workflow execution time for each configuration type

The workflow execution time for the Small configuration i.e. m1.small flavour type is better than the Tiny configuration i.e. m1.tiny flavour type. However, the execution time is higher than the execution time obtained using the Random configuration. The reason for the better workflow execution time for the Random configuration can be understood by analysing the provisioned resources. In the Random configuration, the resources were provisioned randomly, which means that a resource could be provisioned with any of the m1.tiny, m1.small, m1.medium or m1.large configurations. In order to further understand the flavour types of these randomly provisioned resources, the flavour distribution chart is shown in Figure 6.7. This figure shows the types of resources which were provisioned in the Random configuration for each workflow execution. From Figure 6.7, it is clear that all flavour types have been used to provision resources on the Cloud in the Random configuration. Moreover, this figure also shows that more than 50% of the virtual machines were provisioned with the m1.large and m1.medium flavour types, which provide more resources to a virtual machine than the m1.small configuration. Since the jobs have extra resources in terms of the RAM, CPU and Hard disk, this affects their execution times and thus the overall workflow execution time is reduced.

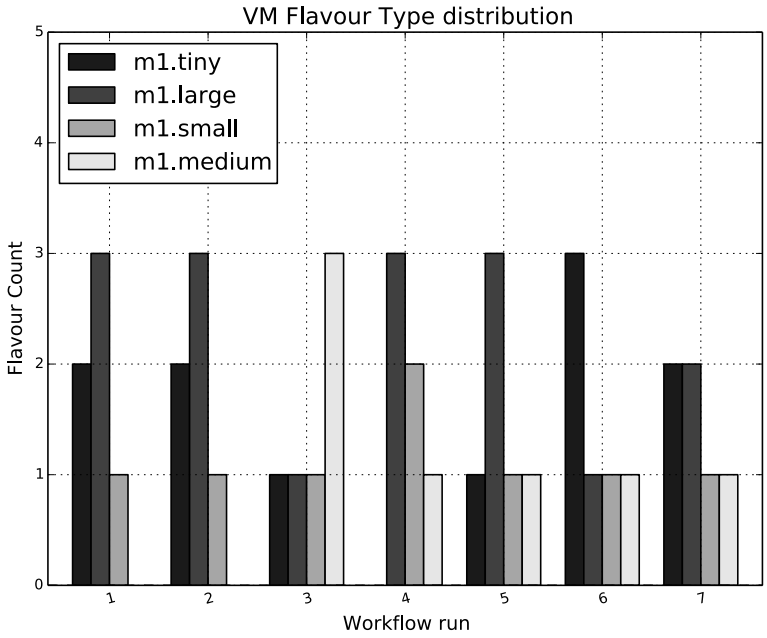


Figure 6.7: Frequency of a flavour type selected randomly during a workflow execution

6.2.4.1 Discussion

With these results, it has been established that the factors related to a Cloud resource identified in this research study (discussed in Section 3.3). They not only affect a job execution performance and its failure rate, but also can affect a workflow execution performance. The proposed system, ReCAP, is capable of capturing these factors in its Cloud-aware provenance. It can be argued that

the scheduler in a workflow management system (WMS) can filter the required resource for a given job in order to avoid job failures, provided the job has announced its required resource configuration. Nonetheless, this information remains useful in acquiring the desired resources from the Cloud instead of acquiring them with random configurations. Moreover, adding these factors in the collected provenance also provides an insight about the types of resources selected by the WMS for job execution. In Section 6.2.4, it was shown that the workflow execution performance improves if the acquired virtual machines have improved resource configurations than the previous execution of the same workflow. However, if the acquired virtual machines do not have better configurations than the original execution, the workflow performance will deteriorate (as shown in Figure 6.6). Therefore, it can be argued that a user can achieve a similar workflow execution performance by provisioning the similar execution infrastructure on the Cloud by using the Cloud-aware provenance information.

Moreover, Section 6.2.3 also discusses the effect of CPU MIPS on workflow execution time. The simulation results confirm that this parameter should be collected in provenance information. Although the current Cloud APIs do not support the access to this information and also Cloud Providers do not widely support this parameter in resource provisioning pipeline, the result in Figures 6.5 and 6.4 show its significance. From the simulation result, it was found that a job's performance was directly related to the MIPS value. As discussed earlier in Section 6.2.2, number of CPUs or CPU speed as MIPs can affect a job performance. This information can be useful for experiments which are both compute and data intensive such as gnomic experiments. In the absence of this information, a compute intensive analysis cannot be reproduced correctly because we cannot use this parameter in while requesting a resource.

6.3 Experiment 2: Mapping Algorithm Experiment

In order to evaluate the different Cloud scenarios (as discussed in Section 4.1) and the proposed job-to-Cloud resource mapping approaches, a set of experiments have been designed and performed. These experiments help in justifying the need for the Eager approach. Since the used workflow management system i.e. Pegasus was designed for the Grid in which resources remain available even after a workflow finishes its execution, it is critical to determine the time that Pegasus takes to store the execution machine information in its database. In this regard, a very basic workflow consisting of a single job has been designed and executed using Pegasus. In order to measure the host information availability time in the Pegasus database, a monitoring thread (discussed in Section 5.3.2.1) periodically (after every two seconds) polls the database for the given job. In Pegasus, a job goes through the following states, which are discussed in order below.

1. $SUBMIT$: This state occurs when a job is submitted to the Condor cluster for execution. However, this does not specifically mean that the job has actually started its execution because

the job could be in the waiting queue and ready to be assigned to a worker node from the Condor cluster.

2. $EXECUTE$: This state occurs when a job is assigned to a worker node and it starts execution.
3. $JOB\_TERMINATED$: This occurs when a job has finished its execution.
4. $JOB\_SUCCESS$: Once a job has finished, its status information from Condor gives an indication about its execution success or failure. If a job has failed, no post script job is executed. Thus, this becomes the last state of a job. In case of a successful execution, a post script job is executed to process the job output (discussed in Point 5).
5. $POST\_SCRIPT\_START$: Once a job has finished successfully, its postscript job is started. This job is responsible for two main tasks. Its first task is to rename the job's stdout and stderr files and attach a postfix of XXX.out and XXX.err where XXX is the retry count, which determines the number of times a job was submitted. Its second task is to parse the job's .out file and to determine the exit code of a job execution.
6. $POST\_SCRIPT\_SUCCESS$: This occurs when the postscript job finishes successfully.
7. $POST\_SCRIPT\_TERMINATED$: This occurs when Pegasus realizes and marks that the post script job has finished.

For this test, the job state $POST\_SCRIPT\_TERMINATED$, which is the last state a job has in Pegasus, is used to determine the delay in registering host information for a job in the database. The host information availability time for a job is calculated by the following:

$$HostAvailabilityTime_{job} = MonitoringTime_{job} - JobFinishTime_{job}$$

In this equation, the $MonitoringTime_{job}$ comes from the monitoring thread when it discovers that job's host information is available in the Pegasus database. The $JobFinishTime_{job}$ comes from the time when the job finishes its final state in Pegasus. Figure 6.8 shows the measured time difference between the time when a workflow job finishes and the time when its host information is available in the Pegasus database. The test workflow was executed 20 times and the average delay to detect host information in the Pegasus database was 27.65 $\pm$ 0.9333 seconds. This means that the Static approach will not be able to determine the host information of a job from the Pegasus database within this time and thus it would not establish the job-to-Cloud resource mapping. In a dynamic Cloud scenario, this is a sufficient time delay in which a virtual resource could be released without a user knowing about it. In this case, the Static mapping approach will not be able to establish job-to-Cloud resource mapping because the Cloud resource will no longer be accessible. To counter this and to support the dynamic Cloud scenario, the Eager mapping approach was presented in Section 4.1.2.

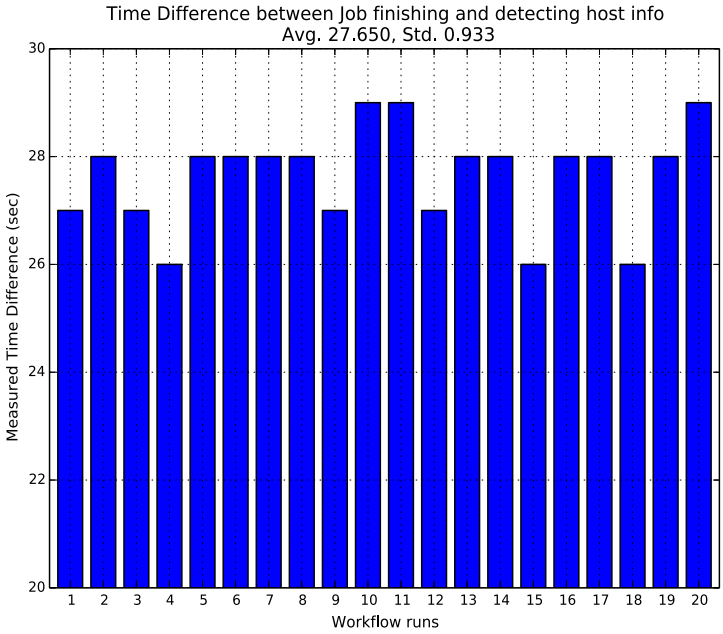


Figure 6.8: Measured Time difference between Job Finish time and Host information availability time

Since the monitoring thread had a sleep interval of 2 seconds, this additional time delay should be eradicated from the result in order to analyse it properly. Figure 6.9 illustrates that the sleep interval has not influenced a great deal in terms of average host information availability time and its standard deviation statistics.

In order to determine the host information availability time using the Eager approach, the same experiment was repeated multiple times. In this test, the host information availability time is calculated by subtracting the job monitoring time, which is obtained from the temporary map table (used by the Eager approach), from the job finish time registered by Pegasus. Figure 6.10 shows that the Eager approach discovers the host information of a job much faster when compared to the time it takes Pegasus to make the host information available in its database. This is because the Eager approach interacts with the Condor cluster as soon as it discovers that a job has been submitted to the Condor for execution. From Condor, the Eager approach discovers the machine information which is then linked to the Cloud infrastructure information to register a temporary mapping. All of these discovery and mapping processes happen during the job execution, which unlike Pegasus that waits for a job to finish. Due to this, the Eager approach discovers the host information of a job much quicker. The reason for the negative values in Figure 6.10 is because the $MonitoringTime_{job}$ happens much earlier than the $JobFinishTime_{job}$.

These results confirm that the Static approach can work in a Static Cloud environment, which is used by Pegasus. However, in dynamic Cloud environment, where a resource can be released once a job is finished, the Static approach would not work. It is because Pegasus can take up to 27 seconds

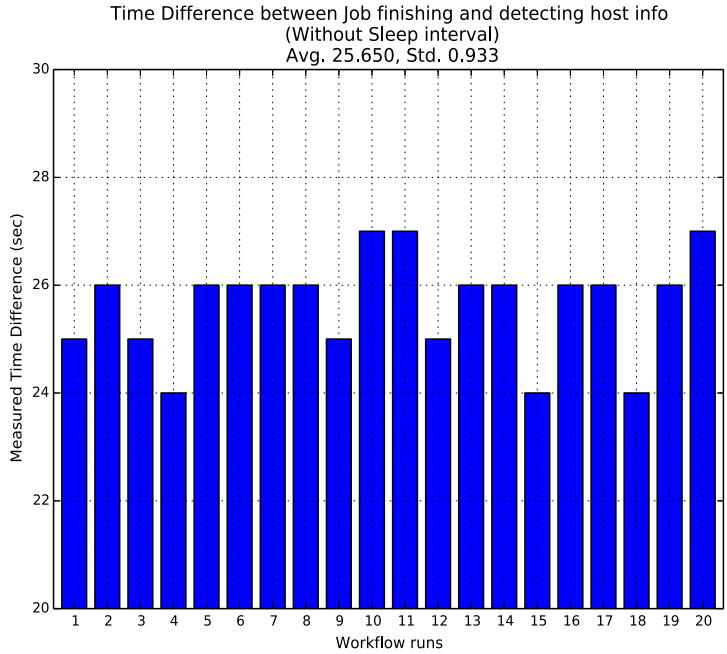


Figure 6.9: Measured Time difference between Job Finish time and Host information availability time without the Sleep delay

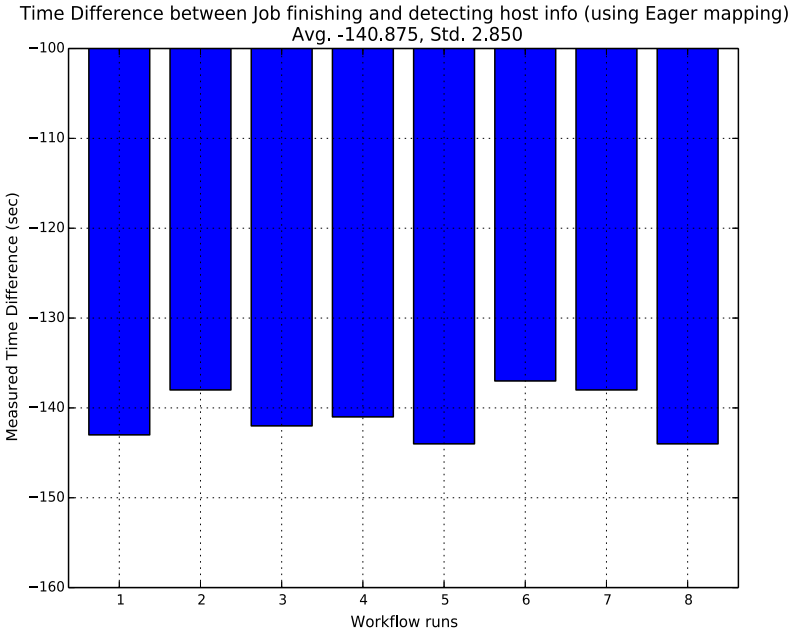


Figure 6.10: Measured Time difference between Pegasus existing and the Eager Approach

to register job's host information in its database. Since the Static approach relies upon the host information from the WMS database, which is Pegasus in this case, it is possible that the resource would not be available on the Cloud. This can lead to incorrect job-to-Cloud resource mapping because the Static approach would not find the resource on the Cloud and thus could not retrieve its configurations. The result in Figure 6.10 confirms the working of the Eager approach which was designed specifically to determine the job-to-Cloud resource mapping as early as possible. This shows that the Eager approach should be used in the dynamic Cloud environment.

6.4 Experiment 3: Provenance Correctness and Completeness

In this section, the correctness and completeness of the captured provenance information is evaluated by using the approaches presented in Section 4.2.2. Since the focus of this research work is on the Cloud information, the provenance correctness analysis focused on the captured Cloud infrastructure details. Based on the presence of Cloud specific parameters in the captured provenance, correctness of the provenance is determined (as previously discussed in Section 4.2.2.1 and Section 3.4. In provenance completeness analysis, it was verified that ReCAP had captured and linked Cloud resource information with all the workflow jobs. This is to verify if there are some jobs without correct Cloud-aware provenance or missing information. Following subsections present and discuss the result of provenance correctness and completeness.

6.4.1 Provenance Correctness Analysis

As discussed in Section 4.2.2.1, the correctness of the collected provenance was measured in this research by evaluating the presence of the Cloud specific parameters i.e. the vCPU, the RAM, the Hard disk, the Flavour name and the Image name in the Cloud-aware provenance. In this regard, three different workflows (shown in Table 6.2) with varying numbers of jobs were executed and their provenance was captured. The *Wordcount* workflow had four jobs, the *pseudo Wordcount* workflow also had four jobs and the *onejob* workflow had only one job. The jobs in the *pseudo Wordcount* workflow did not actually process the data files instead they only mimicked the behaviour of the jobs of the *Wordcount* workflow by adding sleep intervals of the given time, which was passed as an argument. This helps in answer **Research Question 3** (discussed in Section 1.4).

Table 6.2: Workflow used in this experiment

Workflow Type	Number of Jobs
Wordcount	4
pseudo Wordcount	4
onejob	1

The collected provenance was then analysed by comparing it with the original workflow jobs to deduce whether the collected provenance is correct. In doing so, it searched for the Cloud resource

mapping stored against a particular job. If all the identified Cloud parameters were present and none of them were missing, the correctness produced True. It would return False for the job records for which it could not find any of the identified Cloud parameters. Table 6.3 shows the Cloud specific parameters collected in the Cloud-aware provenance for the three workflows. The last column in this table shows the correctness of the record. From the table, it is clear that only those records were determined as correct (highlighted in Green) for which the Cloud specific parameters were all present. In the absence of all or any of these parameters, the mapping information is determined as incorrect (highlighted in Red). The result shown in Table 6.3 verifies the three use cases (discussed in Section 5.3.3) identified in the correctness analysis. This result shows that the provenance correctness analysis worked according to the theoretical understanding (discussed in Sections 3.4 and 4.2.2.1). The purpose of provenance correctness analysis is to ensure that the registered provenance does not miss any of the Cloud resource configuration parameters. In case of missing parameters, it cannot be reliably determined that the infrastructure provisioned to reproduce a workflow is correct.

Table 6.3: Check Correctness of the provenance information especially Cloud parameters

Workflow	JobID	vCPU	RAM (MB)	Hard Disk (GB)	Flavour type	Image Name	Correctness
onejob	3586	1	2048	20	m1.small	uwe-condor-snapshot	True
Wordcount	3429	1	2048	20	m1.small	uwe-condor-vm	True
Wordcount	3430	1	2048	20	m1.small	uwe-condor-snapshot	True
Wordcount	3441	1	2048	20	m1.small	uwe-condor-vm	True
Wordcount	3440	1	2048	20	m1.small	uwe-condor-vm	True
Wordcount	3021	1	2048	20	m1.small	uwe-condor-snapshot	True
Wordcount	3022	1	2048	20	m1.small	uwe-condor-vm	True
Wordcount	3033	1	2048	20	m1.small	uwe-condor-vm	True
Wordcount	3032	1	2048	20	m1.small	-	False
pseudo Wordcount	3705	1	2048	20	m1.small	uwe-condor-snapshot	True
pseudo Wordcount	3703	1	2048	20	m1.small	uwe-condor-vm	True
pseudo Wordcount	3704	-	-	-	-	-	False
pseudo Wordcount	3706	1	2048	20	m1.small	uwe-condor-snapshot	True

6.4.2 Provenance Completeness Analysis

As discussed in Section 4.2.2.2, the completeness of the collected provenance is measured in this research by comparing the workflow provenance with the actual workflow description along with measuring the correctness of the collected provenance. In this regard, three different workflows (shown in Table 6.2) with varying numbers of jobs were executed and their provenance was captured. The *Wordcount* workflow had four jobs, the *pseudo Wordcount* workflow also had four jobs and the *onejob* workflow had only one job. The jobs in *pseudo Wordcount* workflow do not actually process the data files instead they only mimic the behaviour of the jobs of the *Wordcount* workflow by adding sleep intervals of the given time, which is passed as an argument.

The collected provenance is then analysed by comparing it with the original workflow jobs to

Table 6.4: Workflow used in this experiment

Workflow Type	Number of Jobs
Wordcount	4
pseudo Wordcount	4
onejob	1

deduce whether the collected provenance is correct. Table 6.5 shows the Cloud specific parameters collected in the Cloud-aware provenance for the three workflows. The last column in this table shows the correctness of the record. From the table, it is clear that only those records are determined as correct (highlighted in Green) for which the Cloud specific parameters are present. In the absence of all or any of these parameters, the mapping information is determined as incorrect (highlighted in Red). Table 6.5 also shows that the proposed algorithm performed according to the identified two usecases i.e. Use Case 4 and Use Case 5 (discussed in Section 5.3.3).

Table 6.5: Check Completeness of the collected provenance

Workflow	JobID	MapExists	CloudParams	Completeness			
				Actual Jobs	Found Jobs	CScore	Status
onejob	3586	True	True	1	1	1	True
Wordcount	3429	True	True	4	4	4	True
Wordcount	3430	True	True				
Wordcount	3441	True	True				
Wordcount	3440	True	True				
Wordcount	3021	True	True	4	4	3	False
Wordcount	3022	True	True				
Wordcount	3033	True	True				
Wordcount	3032	True	False				
pseudo Wordcount	3705	True	True	4	4	2	False
pseudo Wordcount	3703	True	False				
pseudo Wordcount	3704	False	False				
pseudo Wordcount	3706	True	True				

The $MapExists$ column in Table 6.5 shows the existence of the job-to-Cloud resource mapping for a particular job. The $CloudParams$ column shows that all the identified Cloud parameters for that job-to-Cloud resource mapping are present. The *Actual Jobs* columns shows the actual number of jobs found in the workflow description and the *Found Jobs* shows the number of jobs found in the provenance data for that workflow. The $CScore$ refers to the COMPLETENESS value calculated using Algorithm 7. The completeness analysis ensures that ReCAP has collected and stored Cloud-aware provenance for all the jobs described in the workflow. This helps in confirming that ReCAP would be able to re-provision resources consumed by all the jobs of the given workflow.

6.4.3 Discussion

In this section, the correctness and completeness of the Cloud-aware provenance was discussed. Table 6.3 showed that the proposed algorithm could detect missing parameters in the Cloud-aware provenance of a job. The correctness analysis focuses only on the Cloud parameters and their presence in the provenance. On the other hand, the completeness analysis focuses on all the jobs of a

workflow and their collected correct provenance. Table 6.5 showed that the algorithm could retrieve the job information from the workflow description, and analysed their Cloud-aware provenance. In case of missing job-to-Cloud resource mapping or a difference between actual jobs and found jobs, the completeness analysis fails. It returns True if the Cloud-aware provenance for all the jobs of a workflow are correct.

The purpose of determining the correctness and completeness of the collected provenance is to ensure that the captured provenance is correct. In the absence of Cloud resource configuration for a job or incomplete provenance, ReCAP could not re-provision a resource on the Cloud because Cloud-aware provenance is not complete. This can lead to a resource provisioning with incorrect configuration which can then lead to incorrect infrastructure comparison and also graph structure comparison. As it was discussed in Section 6.2.4 that the Cloud configuration can affect a job and also a workflow performance, thus an incorrect resource configuration can lead to incorrect results. The outcome of completeness analysis is used in determining the workflow reproducibility (discussed later in Section 6.5) as this ensures that all the jobs have correct Cloud-aware provenance in the database. It must be noted that the correctness and completeness analysis works on an individual job of a given workflow and thus cannot compare two workflows. For this, a detailed discussion of comparing two workflow execution provenance is presented in Section 6.6.

6.5 Infrastructure Re-provisioning using ReCAP

In this experiment (previously outlined in Section 5.3.4), three different workflows have been executed. These three workflows are: (1) Wordcount, (2) Montage and (3) ReconAll. These workflows were submitted using Pegasus and their Cloud-aware Provenance was collected. The purpose of this experiment is to evaluate the ability of ReCAP to handle different types of workflows with varying number of jobs and to re-provision resource on the Cloud using the Cloud-aware provenance information. This also evaluates the significance of provisioning similar Cloud infrastructure for workflow re-execution. Following subsections provide the detailed analysis of the results.

6.5.1 Wordcount Workflow

The Wordcount workflow was executed using the Pegasus. The original execution of this workflow was assigned an ID 132. Table 6.6 shows the interlinked provenance mapping (both the workflow provenance and the Cloud infrastructure information), in the ReCAP database for this workflow. The collected information includes the flavour and image (image name and Image id) configuration parameters. The Image id uniquely identifies an OS image hosted on the Cloud and this image contains all the software or libraries used during the job execution. This workflow is then resubmitted by re-provisioning the resources on the Cloud using the Cloud-aware provenance, and its provenance

is captured as shown in Table 6.7. The captured provenance of the reproduced workflow shows that the system was able to re-provision the similar execution resources on the Cloud.

Table 6.6: Cloud-aware Provenance captured for a given Wordcount workflow

WfID	Host IP	nodename	Flavour Id	RAM (MB)	HD (GB)	vCPU	image name	image Id
132	174.16.1.49	uwe-vm3	2	2048	20	1	condorvm-quantal-snapshot	269cfb39-7882-4067-bf20-b3350a4b1b05
132	174.16.1.98	uwe-vm4	2	2048	20	1	condorvm-quantal-snapshot	269cfb39-7882-4067-bf20-b3350a4b1b05

Table 6.7: Provenance data of the reproduced workflow showing that ReCAP successfully re-provisioned similar resources on the Cloud

WfID	Host IP	nodename	Flavour Id	RAM	HD	vCPU	image name	image Id
134	172.16.1.183	uwe-vm4-rep	2	2048	20	1	condorvm-quantal-snapshot	269cfb39-7882-4067-bf20-b3350a4b1b05
134	172.16.1.187	uwe-vm3-rep	2	2048	20	1	condorvm-quantal-snapshot	269cfb39-7882-4067-bf20-b3350a4b1b05

In order to measure the execution time of the original workflow and the reproduced workflow on the similar execution infrastructure on the Cloud, the following mechanism has been adopted. The same workflow was executed five times on the Cloud infrastructure. An average execution time was calculated for these workflow executions and treated as the average execution time of the original workflow. The ReCAP approach was then used to reproduce the same workflow execution by re-provisioning the earlier execution infrastructure using the Cloud-aware provenance. The same workflow was re-executed on the re-provisioned resources to measure the execution time of the reproduced workflow. Figures 6.11 and 6.12 show the workflow execution times for both the original and reproduced workflows respectively.

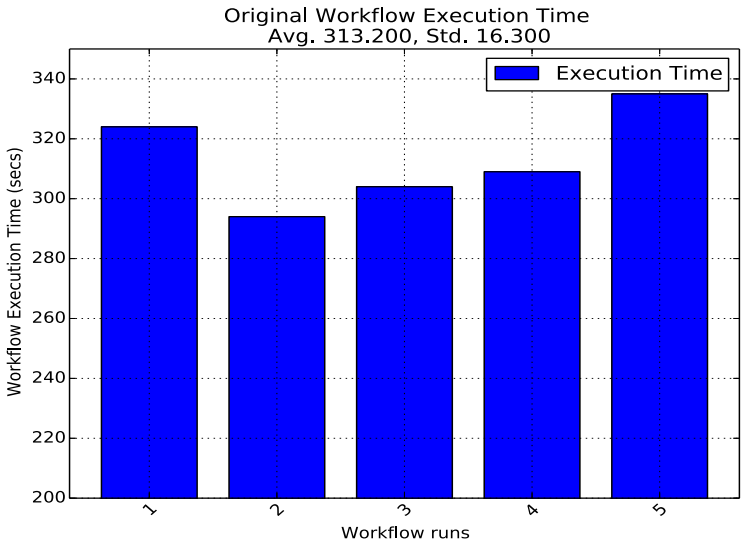


Figure 6.11: Average workflow execution time for the original workflow execution

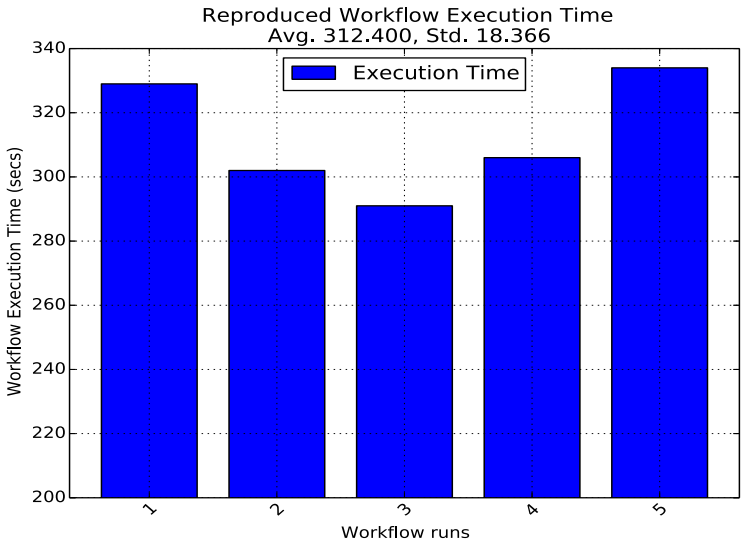


Figure 6.12: Average workflow execution time for the reproduced workflow execution

The average workflow execution time for the original workflow was 313.20 ± 16.30 seconds and the average workflow execution time for the reproduced workflow on the similar execution infrastructure was 312.400 ± 18.366 seconds. This shows that the execution time for the reproduced workflow is almost similar to the original workflow execution time. This result shows that we can expect similar workflow execution performance provided similar execution infrastructure was provisioned. The combined results of both original and reproduced workflow executions are shown in Figure 6.13.

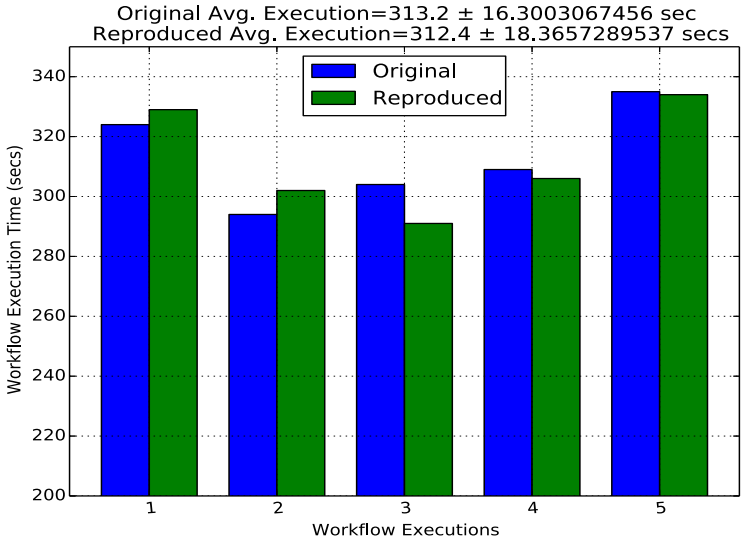


Figure 6.13: Combined result showing the original and reproduced workflow execution times

6.5.2 Montage Workflow

The original execution of this workflow was assigned an ID 533 and the reproduced workflow was assigned an ID 534. Following tables 6.8 and 6.9 show the Cloud infrastructure provisioned for the execution of workflows 533 and 534 respectively. The workflow execution times for the original workflow 533 and reproduced workflow 534 were 285 and 281 seconds respectively. Table 6.9 shows that the Montage workflow execution was reproduced with a similar execution performance on the same Cloud resources provisioned by ReCAP.

Table 6.8: Infrastructure detail captured for the Montage workflow (wfID=533) using ReCAP

WfID	nodename	Flavour Id	RAM	HD	vCPU	image name	image Id
533	uwe-vm3	2	2048	20	1	wf_peg_repeat	f102960c-557c-4253-8277-2df5ffe3c169
533	mon1	2	2048	20	1	montage-condor-setup	2d9787e4-b0e9-4802-bf4f-4a0c868bb11a

Table 6.9: Infrastructure detail captured for the reproduced Montage workflow (wfID=534) using ReCAP

WfID	nodename	Flavour Id	RAM	HD	vCPU	image name	image Id
534	mon1-rep	2	2048	20	1	montage-condor-setup	2d9787e4-b0e9-4802-bf4f-4a0c868bb11a
534	uwe-vm3-rep	2	2048	20	1	wf_peg_repeat	f102960c-557c-4253-8277-2df5ffe3c169

6.5.3 ReconAll Workflow

The ReconAll workflow consists of one job script only and requires one input file. Its execution requires one virtual machine on the Cloud. The original execution of this workflow was assigned ID 545 and its captured Cloud-aware provenance using ReCAP is shown in Table 6.10. The same workflow was re-executed using ReCAP and its provenance information is recaptured. The reproduced workflow execution was assigned an ID 547. The recaptured provenance of the reproduced workflow 547 is show in Table 6.11. These tables show that ReCAP was able to capture the Cloud infrastructure information in the original execution and was also able to re-provision the resource with the same configurations on the Cloud. The execution time of the original workflow was 32599.16 ± 158.73 seconds and the execution time of the reproduced workflow was 32620.0 ± 147.78 seconds.

Table 6.10: Infrastructure detail captured for the ReconAll workflow (wfID=545) using ReCAP

WfID	nodename	Flavour Id	RAM	HD	vCPU	image name	image Id
545	freesurf	2	2048	20	1	freesurf-condor	2ee3c500-61b5-4592-8d54-e572536b5df1

Table 6.11: Infrastructure detail captured for the ReconAll reproduced workflow (wfID=547) using ReCAP

WfID	nodename	Flavour Id	RAM	HD	vCPU	image name	image Id
547	freesurf-rep	2	2048	20	1	freesurf-condor	2ee3c500-61b5-4592-8d54-e572536b5df1

In order to see the effect of resource configurations on the job execution performance, and subsequently on the workflow execution time, the same workflow was re-executed on a resource with different configuration (see Table 6.12). This time the resource was provisioned manually but its provenance was captured through ReCAP. The workflow execution time in this case was 32278.5 seconds. From this result, it must be noted that the execution time decreases on a resource with better resource configurations i.e. more RAM and CPUs. This result verifies the original argument that resource configurations can affect a job execution and thus overall workflow execution performance. This is the reason why capturing resource configurations for workflows execution on the Cloud is essential.

Table 6.12: Infrastructure details captured for the workflow (wfID=554) using ReCAP

WfID	nodename	Flavour Id	RAM	HD	vCPU	image name	image Id
554	newfree	3	4096	40	2	freesurf-condor	2ee3c500-61b5-4592-8d54-e572536b5df1

6.5.4 Discussion

In this section, the execution of three different workflows has been discussed and the ReCAP's ability to capture job-to-Cloud mapping has been presented. These results also show that ReCAP can process and interlink various workflow types of varying number of jobs. Tables 6.6 and 6.7 show that ReCAP can successfully captured the Cloud provenance and interlink it with the workflow provenance. Using the Cloud-aware provenance, ReCAP can also re-provision the Cloud resources with same configurations on the Cloud for workflow re-execution. ReCAP framework has been tested with the Montage workflow with 35 jobs and results show that the proposed system can capture and interlink its provenance with the Cloud information (see Tables 6.8 and 6.9). However, some variation in the workflow execution time was also found. This variation is largely caused by the delays incurred by the workflow management system and also by accessing and storing input and output files.

In this experiment, the ReconAll workflow, which is used in the N4U project, has also been executed and its provenance information is captured using ReCAP. The provenance information for both executions i.e. original and reproduced shown in Tables 6.10 and 6.11) is similar. In order to determine the effect of resource configuration on execution, ReconAll workflow was executed on different resource configuration (see Table 6.12). This showed that the workflow execution was significantly affected by the change in the resource configuration. In case of workflow 554, the workflow execution time decreases by 253 seconds as compared to the original execution workflow 545. This result further strengthens the claim that workflow execution performance can be affected by Cloud resource configurations. Therefore, this information along with the workflow provenance was captured in this research study. The absence of this information can affect in two ways. Firstly, resource provisioning on the Cloud without these parameters is not possible. Secondly, randomly

provisioning the resources on the Cloud can effect on the results which can affect negatively on the reproducibility.

In this section, it has been presented and analysed that ReCAP can capture the Cloud infrastructure information and interlink it with the workflow provenance. ReCAP can also use this information while reproducing a workflow re-execution by re-provisioning the Cloud resources with same configurations. Moreover, results have shown that we can expect similar execution performance provided similar execution infrastructure is used. This finding is particularly important for workflows or analysis developers who are keen about performance. But only these points are not enough in order to analyse the workflow reproducibility. According to the provenance comparison point (discussed in Section 3.3), a detailed analysis of the provenance graphs of original and reproduced workflows is presented in Section 6.6.

6.6 Provenance Comparison Analysis

As discussed in Section 3.3, the provenance comparison should be performed at three levels to determine workflow reproducibility. These three levels are; (1) structural analysis, (2) infrastructure analysis and (3) output analysis. In order to perform the structural analysis of the two given provenance traces two algorithms i.e. Boolean comparison (discussed in Section 4.2.1.2) and Numerical comparison (discussed in Section 4.2.1.3). The Boolean comparison provides a result in True or False and the Numerical comparison provides the degree of similarity between the two given provenance graphs. Following subsections provide analysis of the results in detail.

6.6.1 Structure Analysis

For the above case, both workflows i.e. 132 and 134 have been executed by provisioning the same virtual machines on the Cloud. The graph representation of them is shown in Appendix B. The Boolean Comparison produces $True$ because both graphs contain equal numbers of nodes (i.e. job, virtual machines, files) and all the edges between them are also the same. The structural similarity between the two graphs calculated by the Numerical Comparison algorithm is also 100%. This shows that both graphs are the same and the jobs were executed on the same virtual machines. The infrastructure analysis on both the provenance traces shows that two virtual machines with same configurations were provisioned to execute workflow 134.

However, it is possible that the structural analysis of two given provenance graphs of same workflow produces $False$ or $<$ 100% similarity (see Table 6.13). This can be caused by multiple reasons. One of them is a change in the workflow description. The other pertinent reason is a change in the infrastructure information i.e. CAP in the collected provenance. This change in CAP can appear by

provisioning virtual machines with different configurations (see comparison between 132 and 161). Another reason for this change is because of the job scheduling mechanism used by WMS. Since WMS does not incorporate the CAP information in job scheduling decision, it is possible that a job may be submitted to a different machine. These behaviours can be seen in Table 6.13.

Table 6.13: Provenance structural analysis of workflows (132, 134), (132, 161), (161, 165), (316, 207)

Workflow Type	Comparison	Structural Comparison	
		Boolean	Numeric
Wordcount	(132, 134)	True	100%
	(132, 161)	False	92.287%
	(161, 165)	True	100%
	(316, 207)	False	96.51%
Montage	(533, 534)	True	97.66%
	(533, 535)	False	48.7%
ReconAll	(545, 547)	True	100%
	(545, 554)	False	25%

6.6.2 Infrastructure Analysis

This section discusses the infrastructure information captured by the ReCAP framework during workflow execution. By analysing the infrastructure details, the comparison provided in Table 6.13 can be better understood. For this analysis, four workflow executions with workflow ID 132, 161, 316 and 207 are selected because their structural analysis is False and below 100%. The infrastructure detail of workflow 165 is also used to determine the reason behind 100% structural similarity. Table 6.14 shows the infrastructure information captured for these workflows.

Table 6.14: Infrastructure details captured for the workflows using ReCAP

WfID	nodename	vCPU	HD	RAM	Flavour Id	image name	image Id
132	uwe-vm9	1	20	2048	2	condorvm-quantal-snapshot	269cfb39-7882-4067-bf20-b3350a4b1b05
132	uwe-vm8	1	20	2048	2	condorvm-quantal-snapshot	269cfb39-7882-4067-bf20-b3350a4b1b05
161	uwe-vm4	1	20	2048	2	uwe-condor-snapshot	4d05e308-b46d-4a9e-8a29-237bebf11161
161	uwe-vm3	1	20	2048	2	uwe-condor-vm	ca16f4a2-dba3-44c0-a3b5-72267094ff40
161	uwe-vm4	1	20	2048	2	uwe-condor-snapshot	4d05e308-b46d-4a9e-8a29-237bebf11161
161	uwe-vm3	1	20	2048	2	uwe-condor-vm	ca16f4a2-dba3-44c0-a3b5-72267094ff40
165	uwe-vm3-rep	1	20	2048	2	uwe-condor-vm	ca16f4a2-dba3-44c0-a3b5-72267094ff40
165	uwe-vm4-rep	1	20	2048	2	uwe-condor-snapshot	4d05e308-b46d-4a9e-8a29-237bebf11161
316	uwe-vm4	1	20	2048	2	uwe-condor-snapshot	4d05e308-b46d-4a9e-8a29-237bebf11161
316	uwe-vm3	1	20	2048	2	uwe-condor-vm	ca16f4a2-dba3-44c0-a3b5-72267094ff40
207	uwe-vm3-rep	1	20	2048	2	uwe-condor-vm	ca16f4a2-dba3-44c0-a3b5-72267094ff40
207	uwe-vm4-rep	1	20	2048	2	uwe-condor-snapshot	4d05e308-b46d-4a9e-8a29-237bebf11161

As it can be seen that the configurations for the provisioned virtual machines for workflows 161 and 165, and 316 and 207 are same, however the structural similarity is different for (161, 165) and (316, 207) comparisons. As discussed in previous Section 6.6.1, one of the reasons for this

Table 6.15: Virtual machines used by the jobs of workflow 161

WfID	Job Name	Machine	RAM	HD	vCPU	Flavour Id	image name	image Id
161	analyse_ID0000002	uwe-vm4	2048	20	1	2	uwe-condor-snapshot	4d05e308-b46d-4a9e-8a29-237bebf11161
161	analyse_ID0000003	uwe-vm3	2048	20	1	2	uwe-condor-vm	ca16f4a2-dba3-44c0-a3b5-72267094ff40
161	preprocess_ID0000001	uwe-vm4	2048	20	1	2	uwe-condor-snapshot	4d05e308-b46d-4a9e-8a29-237bebf11161
161	merge_ID0000004	uwe-vm3	2048	20	1	2	uwe-condor-vm	ca16f4a2-dba3-44c0-a3b5-72267094ff40

Table 6.16: Virtual machines used by the jobs of workflow 165

WfID	Job Name	Machine	RAM	HD	vCPU	Flavour Id	image name	image Id
165	analyse_ID0000002	uwe-vm4	2048	20	1	2	uwe-condor-snapshot	4d05e308-b46d-4a9e-8a29-237bebf11161
165	analyse_ID0000003	uwe-vm3	2048	20	1	2	uwe-condor-vm	ca16f4a2-dba3-44c0-a3b5-72267094ff40
165	preprocess_ID0000001	uwe-vm4	2048	20	1	2	uwe-condor-snapshot	4d05e308-b46d-4a9e-8a29-237bebf11161
165	merge_ID0000004	uwe-vm3	2048	20	1	2	uwe-condor-vm	ca16f4a2-dba3-44c0-a3b5-72267094ff40

difference is the job scheduling. To verify this, the following tables 6.15, 6.16, 6.17 and 6.18 show the job-to-Cloud resource mapping for each job of workflows 161, 165, 316 and 207 respectively.

From the tables 6.15 and 6.16, it is clear that all jobs of workflow 165 have been executed on the same virtual machines as was the case with the jobs of workflow 161. Due to this reason, there is no edge difference found between their provenance graphs and thus the resultant structural similarity was 100%. However, this was not the case with workflows 316 and 207 as shown in the Tables 6.17 and 6.18. The differences between jobs scheduled to different machines are highlighted in red.

Table 6.17: Virtual machines used by the jobs of workflow 316

WfID	Job Name	Machine	RAM	HD	vCPU	Flavour Id	image name	image Id
316	analyse_ID0000002	uwe-vm3	2048	20	1	2	uwe-condor-vm	ca16f4a2-dba3-44c0-a3b5-72267094ff40
316	analyse_ID0000003	uwe-vm4	2048	20	1	2	uwe-condor-snapshot	4d05e308-b46d-4a9e-8a29-237bebf11161
316	preprocess_ID0000001	uwe-vm3	2048	20	1	2	uwe-condor-vm	ca16f4a2-dba3-44c0-a3b5-72267094ff40
316	merge_ID0000004	uwe-vm3	2048	20	1	2	uwe-condor-vm	ca16f4a2-dba3-44c0-a3b5-72267094ff40

Table 6.18: Virtual machines used by the jobs of workflow 207. Rows in red highlights the changes due to scheduling.

WfID	Job Name	Machine	RAM	HD	vCPU	Flavour Id	image name	image Id
207	analyse_ID0000002	uwe-vm4	2048	20	1	2	uwe-condor-snapshot	4d05e308-b46d-4a9e-8a29-237bebf11161
207	analyse_ID0000003	uwe-vm3	2048	20	1	2	uwe-condor-vm	ca16f4a2-dba3-44c0-a3b5-72267094ff40
207	preprocess_ID0000001	uwe-vm4	2048	20	1	2	uwe-condor-snapshot	4d05e308-b46d-4a9e-8a29-237bebf11161
207	merge_ID0000004	uwe-vm3	2048	20	1	2	uwe-condor-vm	ca16f4a2-dba3-44c0-a3b5-72267094ff40

Table 6.14 shows that the configurations of the provisioned resources are the same. However, there is a change in the job-to-Cloud resource mapping. Many of the virtual machine configurations such as are same for the jobs of both workflows 316 and 207 (see Tables 6.17 and 6.18). However, the OS image i.e. image name and image id columns are different. For example, the job $analysis\_ID1000002$ of workflow 207 was executed on OS image $uwe - condor - snapshot$, however the same job of workflow 316 was executed on virtual machine with OS image $uwe -$

$condor - vm$. Due to this change in the resource configuration, the edges between the job and the VM nodes are changed and thus affecting the structural similarity analysis. These two results show that the proposed structural analysis algorithms can detect a slight change in the resource configurations while comparing two provenance graphs. This also shows that job scheduling can impact the structural comparison even if the re-provisioned infrastructure is same. Since OS image provides a complete software stack including libraries and binaries, a change in OS image can possibly result into a failed or ill-performed job execution. Gronenschild et al. 2012 has also raised the same concern. Based on this analysis, it can be proposed that job scheduling algorithm should also incorporate the Cloud-aware provenance information.

6.6.3 Workflow Output Analysis

The other aspect in evaluating the workflow reproducibility is to verify the outputs produced by both workflows (as discussed in Section 3.3). This has been achieved using the algorithm presented in Section 4.2.3. Four jobs in both the given Wordcount workflows i.e. 132 and 134 produce the same number of output files (see Table 6.19). The Split job produces two output files i.e. wordlist1 and wordlist2. Two analysis jobs, Analysis1 and Analysis2, consume the wordlist1 and wordlist2 files, and produce the analysis1 and analysis2 files respectively. The merge job consumes the analysis1 and analysis2 files and produces the merge_output file. The hash values of these files are shown in the MD5 Hash column of the Table 6.19; of here both given workflows are compared with each other. For instance, the hash value of wordlist1 produced by the Split job of workflow 134 is compared with the hash value of wordlist1 produced by the Split job of workflow 132. If both the hash values are same, the algorithm will return true. This process is repeated for all the files produced by both workflows. The algorithm confirms the verification of workflow outputs if the corresponding files in both workflows have the same hash values. Otherwise, the verification process fails, which means that both the workflows have not produced the same output. Consequently, this means that the workflow result could not be reproduced.

Similar is the case with the Montage workflow. This workflow produces and registers four output files. Two files are produced by the mAdd job, one file is produced by the mShrink job and the final output (mosaic) file is produced by the mJPEG job. The other intermediary files produced and consumed during the workflow execution are not registered as output files. Table 6.20 shows the outputs produced by the workflow 533 and 534. Seeing the MD5 Hash column, it can be deduced that all files produced by these jobs are identical in both workflow executions. The actual mosaic output files produced by these workflow executions are shown in Appendix B.3.

However, the proposed approach to compare workflow outputs using MD5 hash was not fully applicable to the ReconAll workflow. The output of FreeSurfer analysis depends on the Operating System, FreeSurfer version and OS libraries (Glatard et al. 2015). Although the virtual machines provisioned using the *freesurfer-condor* image contains the same OS, libraries and FreeSurfer version,

Table 6.19: Comparing outputs produced by workflows 132 (original workflow) and 134 (reproduced workflow)

Job	WF ID	Cloud Container Name	File Name	MD5 Hash
Split	132	wfoutput30042014123011	wordlist1	0d934584cbc124eed93c4464ab178a5d
	134	wfoutput30042014125819	wordlist1	0d934584cbc124eed93c4464ab178a5d
	132	wfoutput30042014123011	wordlist2	1bc6ffead85bd62b5a7a1be1dc672006
	134	wfoutput30042014125819	wordlist2	1bc6ffead85bd62b5a7a1be1dc672006
Analysis1	132	wfoutput30042014123011	analysis1	494f24e426dba5cc1ce9a132d50ccbda
	134	wfoutput30042014125819	analysis1	494f24e426dba5cc1ce9a132d50ccbda
Analysis2	132	wfoutput30042014123011	analysis2	127e8dbd6beffdd2e9dfed79d46e1ebc
	134	wfoutput30042014125819	analysis2	127e8dbd6beffdd2e9dfed79d46e1ebc
Merge	132	wfoutput30042014123011	merge_output	d0bd408843b90e36eb8126b397c6efed
	134	wfoutput30042014125819	merge_output	d0bd408843b90e36eb8126b397c6efed

Table 6.20: Comparing outputs produced by Montage workflows 533 (original workflow) and 534 (reproduced workflow)

Job	WF ID	Cloud Container Name	File Name	MD5 Hash
mAdd	533	wfoutput21092015082616	mosaic_20150713_012457_17045.fits	d7e28c00577c6a5e145bf1d478be600b
	534	wfoutput21092015091503	mosaic_20150713_012457_17045.fits	d7e28c00577c6a5e145bf1d478be600b
	533	wfoutput21092015082616	mosaic_20150713_012457_17045_area.fits	11da235545a86d08a3e00ff2c84a31bd
	534	wfoutput21092015091503	mosaic_20150713_012457_17045_area.fits	11da235545a86d08a3e00ff2c84a31bd
mShrink	533	wfoutput21092015082616	shrunken_20150713_012457_17045.fits	7c81dc2d39fa1bcb2ba829ff8e650559
	534	wfoutput21092015091503	shrunken_20150713_012457_17045.fits	7c81dc2d39fa1bcb2ba829ff8e650559
mJPEG	533	wfoutput21092015082616	shrunken_20150713_012457_17045.jpg	2e30b4094651e6e1a82d12eb83be1372
	534	wfoutput21092015091503	shrunken_20150713_012457_17045.jpg	2e30b4094651e6e1a82d12eb83be1372

still the job logs contained the timestamps which were different for multiple runs. Moreover, a few job outputs also contained the machine information which could also be different especially if machine name or IP information is logged. Some log file snippets are provided in Appendix B.2. For instance, the file *recon-all.env* produced by workflow 545 generated a different hash as compared to the same file produced by workflow 547 because they had different timestamps. Due to this reason, the MD5 applied on the files produced different results for different workflow executions. A thorough analysis of the produced output files revealed that the main files related to thickness, area and volume of the given brain-scan produced the same hash values in the original and also in the reproduced workflow (see Table B.1). This specific result leads to a conclusion that MD5 approach for comparing workflow outputs works fine in cases when jobs do not include timestamps or machine information in the produced output. Otherwise this approach cannot work. Similar concerns in comparing outputs have been raised in (Missier et al. 2013). In order to overcome this issue in workflow output comparison, the workflow or application developer e.g. the ReconAll developer in the case of ReconAll workflow, should provide a comparator tool. This tool can then be integrated into the output comparison algorithm used for comparing the outputs for that particular workflow.

6.6.4 Discussion

In this section, a detailed analysis of the provenance graph comparison is presented. In doing so, three different aspects of the provenance graph are evaluated and analysed. The structural comparison between two provenance graphs were performed by employing the two proposed comparison algorithms i.e. the Boolean Comparison and Numerical Comparison. From the Boolean Comparison, one can access whether the provided graphs are identical or not. However, it does not provide information about the degree of similarity. For this, Numerical comparison algorithm was applied. From the result (see Table 6.13), it was found that the provenance graphs might not be identical despite the same execution infrastructure used for workflow re-execution. A detailed analysis of the result showed that it was due to two main reasons. First reason was that a change in the workflow itself could affect the graph comparison. The second reason was that a change in the job-to-Cloud resource mapping can also affect the graph structure comparison. From the detailed analysis of the infrastructure information i.e. job-to-Cloud resource mapping, the second case was found when comparing workflow 316 with 207. Tables 6.17 and 6.18 showed that jobs of workflow 207 were executed on different machines when compared against the jobs resource mapping of workflow 316. As discussed earlier, this was due to the job scheduling algorithm in the workflow management system, which is outside the scope of this research work. Since the job scheduling algorithm does not utilize the Cloud-aware provenance information, it schedules job to any of the available resource according to its scheduling policy.

Besides comparing the workflow execution time to determine workflow reproducibility, work-

flow output comparison has also been identified as a means to verify the reproducibility of a workflow execution. Since all workflows in modern scientific experiments are data oriented and they consumed and produced data, this comparison provides a mechanism to verify the produced outputs. The output comparison of the $Wordcount$ workflows showed that the outputs were successfully produced in reproduced workflow. Since the execution time of both the $Wordcount$ workflows i.e. 132 and 134 executing on the same Cloud resources was similar with some variation and produced same outputs, it can be deduced that the $Wordcount$ workflow has been successfully reproduced using ReCAP. Similar was the case with the Montage workflow that produced four outputs. By comparing the outputs of the original workflow execution with the reproduced workflow execution using Workflow Output Comparison Algorithm (as discussed in Section 4.2.3), it was shown that they were identical. Since the execution time of both the Montage workflows executing on the same Cloud resources was similar with some variation, it can be deduced that the Montage workflow has been successfully reproduced using ReCAP. However, the structural analysis of Montage workflows showed that they were not 100% similar despite same execution resources on the Cloud. It was due to the job scheduling as discussed earlier.

In doing output comparison, it was found that applying MD5 hash to compare workflow outputs was not applicable to all workflow outputs. After analysing the results of ReconAll workflow, it was found that a few output files, which were mainly log files, in both workflows did not produce the same hash value. Further analysis of its outputs revealed that the timestamps and machine information were added to the outputs (see Appendix B.2). Since each execution did not happen exactly at the same time, thus a change in timestamps or machine information changes the entire hash value computed using MD5. However, the actual files related to the brain thickness, area, volume and statistics were identical (see Table B.4). From this result, it was inferred that the MD5 approach used in the proposed comparison algorithm worked fine with the job outputs which did not contain timestamps or other machine specific information. The analysis of this result showed that the output comparison utility should be provided by the workflow or application developer.

6.7 Workflow Reproducibility using ReCAP

In order to reproduce a workflow execution on Cloud, three main factors related to workflow execution were identified in Section 3.3. These points include the structure of a workflow, infrastructure used to execute a workflow and the output produced by a workflow. By combining all these together, one can assess the achieved reproducibility of the workflow. In this section, all these three factors are combined (shown in Table 6.21). The values for these factors were obtained from previous results discussed in Section 6.6.

For the Wordcount workflows 132 and 134, the structural similarity was 100% (see Table 6.13) and both were executed on the same Cloud resources. From Table 6.19, it was found that all the

produced outputs by both workflows were identical. Since the reproduced workflow 134 was 100% similar to 132 and executed on same infrastructure producing identical output files with similar execution performance, it can be deduced the execution of workflow 134 is successfully reproduced. Similarly, the re-executed Montage workflow i.e. wf ID 534 was successfully reproduced because it produced the same outputs and also used the same execution infrastructure. Its structural similarity with the original workflow (wfID 533) was not 100%; this was due to the job scheduling in the workflow management system. Similarly, the ReconAll workflow i.e. wf ID 547 was successfully reproduced because it was executed on the same execution infrastructure as was the case with its original execution i.e. wf ID 545. Moreover, its provenance comparison also produced 100% structural similarity. Although some of its files did not produce the same hash values in output comparison, however, the actual files concerned with brain thickness, area, volume etc. and the statistical files were identical. This is why, its output was considered as reproduced. These results demonstrate that ReCAP can aid in achieving workflow reproducing using the Cloud-aware provenance information.

Table 6.21: Comparing different aspects of provenance to determine workflow reproducibility

Workflow Type	Workflows	Completenes Analysis	Structural Analysis	Infrastructure	Output	Reducibility Status
Wordcount	(132, 134)	True	100%	True	True	True
	(132, 161)	True	92.287%	True	True	True
Montage	(533, 534)	True	97.66%	True	True	True
	(533, 535)	True	48.7%	False	True	False
ReconAll	(545, 547)	True	100%	True	True	True

6.8 Provenance Overhead Analysis

In order to evaluate the impact of the proposed provenance capture i.e. the mapping approaches on the workflow execution performance, a modified Wordcount workflow was executed with different provenance capturing approaches. This workflow also has the same four jobs (shown in Figure 5.3) but the jobs in this workflow sleep, to mimic the job processing time instead of actually processing the data, for a given time period, that was passed as an argument to them. This was done in order to measure the impact of only provenance mapping approaches on the workflow execution performance by eliminating the effect of the data transfer time on the workflow execution time. The *preprocess* job sleeps for 120 seconds, the *analysis1* job sleeps for 120 seconds, the *analysis2* job sleeps for 60 seconds and the *merge* job sleeps for 60 seconds. The workflow was executed with three different provenance mapping approaches proposed in this work. In order to evaluate both static and dynamic mapping approaches, both the Static mapping approach and the Eager mapping approach was applied. The third mapping approach i.e. the SNoHi Mapping approach (discussed in Section 4.1.4) was also tested to analyse its impact on the workflow execution performance. For the third mapping approach, the workflow jobs were further modified to collect the machine information during the job

execution. The workflow was executed 10 times for each mapping approach. The workflow execution performance in the presence of these mapping approaches was compared against the workflow execution performance in the absence of these mapping approaches.

The results shown in Figure 6.14 confirms the theoretical understanding of the proposed mapping approaches (as discussed in Section 5.3.6) because it complies with the intended outcomes. As it can be seen there is no great difference in the workflow execution times in the presence of the Static and Eager mapping approaches. The average workflow execution time in the absence of any provenance approach is 434.67 ± 6.52 seconds. The average workflow execution time with the Static Mapping approach is 434.9 ± 4.29 seconds and the average workflow execution under the Eager approach is 434.78 ± 4.68 seconds. The main reason for these mapping approaches not having a major impact on the workflow execution time is because the proposed mapping approaches work outside the virtual machines, thus they don't interfere with the job execution. However, this is not the case with the SNoHi mapping approach in which the provenance information was captured within the job itself. Due to this, there is a small increase in the workflow execution time for the SNoHi mapping approach. The average workflow execution time for the SNoHi mapping approach is 435.56 ± 4.83 seconds. It was expected that the workflow execution time under the SNoHi mapping approach would increase because an additional provenance collection process runs within the job, which increases the job execution time, hence it affects the overall workflow execution time. However, there is no significance increase in the workflow execution time observed with the SNoHi mapping approach. This is because the provenance collection process during the job execution took only 0.0418 ± 0.0017 seconds (on average).

Figure 6.14: Effect of different provenance collection approaches on the workflow execution performance

As discussed in Section 5.3.6, the overhead of disk space usage caused by storing the Cloud-aware provenance information has also been calculated. At the time of writing, there were a total of 77 workflows of different types (i.e. the *Wordcount* workflow, the *Montage* workflow, the *ReconAll* workflow, the *pseudo Wordcount* workflow and the workflow with only one job), which were executed and their mapping information was stored. In order to calculate the average disk space (i.e., the size overhead) required to store one CAP record, only three tables from the database schema were selected i.e. WfCloudMapping, CPUSpecs and CloudFileCatalog. As discussed previously in Section 3.8.9, these three tables helped in storing the job-to-Cloud resource mapping, the resource CPU information including the CPU MIPS, and the location and the metadata of the files produced by the workflow job. The average size overhead to store the job-to-Cloud mapping along with the CPU specifications in the proposed approach amounted to 505.04 bytes. The average size overhead increases to 925.14 bytes after including the information of the file stored on the Cloud. Table 6.22 shows the table sizes, the number of records in each table and the average disk size per record. With this additional size overhead, ReCAP provides the information about the configurations of the Cloud resources used during the workflow execution.

Table 6.22: Average record size to store Cloud-aware provenance

Table Name	Total Size (bytes)	Total records	Avg. size per record (bytes)
WfCloudMapping	475136	1442	329.50
CPUSpecs	49152	280	175.54
CloudFileCatalog	16384	39	420.10
WfSource	1589248.00	77	20,639.58

In order to support the workflow reproducibility on the Cloud by re-executing the original workflow, the ReCAP database schema also supports storing the original files such as the workflow description in the DAX format and the associated configuration files. The average size required to store source files of a workflow is 20,639 bytes which is quite high. The average size can increase as it depends upon the contents of the stored files. This is due to the fact that ReCAP stores the workflow file contents instead of the file locations. Although this aids in retrieving data directly from the database instead of reading from or writing to local filesystem or to a storage service, this size overhead can be avoided by storing only the location of the workflow configuration files and by providing the support for retrieving these files from the given location.

6.8.1 Discussion

This section has presented the results for performance and disk space usage overhead caused by the approaches proposed in ReCAP. While measuring the performance overhead, it was found that the mapping approaches such as the Static, Eager and Lazy did not affect a great deal on the workflow execution. This was due to the fact they did not rely upon any other process or daemon continuously running inside a virtual machine. Since these approaches work outside the virtual machines,

consequently do not overload the machine executing a job. Therefore, a job execution was not affected by them. The size overhead caused by ReCAP to record Cloud-aware provenance is less than 1 KB. Using this much space, ReCAP can store Cloud-aware provenance which later is used to re-provision resources on the Cloud for workflow reproducibility.

6.9 Summary

This chapter presented and discussed the results of different types of experiments which were conducted in order to evaluate the proposed solution and the research study. The first set of results evaluated the significance of the identified Cloud parameters i.e. the RAM, CPU and resource configurations. Figure 6.1 showed that the job failure rate increases in the absence of the required memory. Figure 6.2 showed that the job execution time improves in the presence of better resource configuration. Although these results do not provide new findings, they still serve the purpose of verifying the arguments presented in favour of collecting such Cloud parameters. It was also shown in Figure 6.3 that the compute intensive jobs should be parallelized in order to achieve better execution performance on the Cloud resource offering multiple CPU cores.

In Section 6.2.3, the significance of capturing the CPU MIPS information has been demonstrated. Figures 6.4 and 6.5 showed that workflow execution improves on the resources with higher MIPS value. The current Cloud API limits a user to request a Cloud resource based on this parameter. The existing Cloud API remains at the number of CPU level when provisioning a Cloud resource, and does not support the CPU specification level e.g. the CPU speed or MIPS information. However, the result suggested that this information should be captured in the provenance. It is argued and proposed that this information should be made available to the user in acquiring resources from the Cloud. Moreover, this provenance data can also be used to look in depth of the machine configuration and to use it when sharing or discussing results with the fellow researchers. Following this, the chapter also presented the workflow performance (shown in Figure 6.6) on different Cloud resource configurations. It was found that the workflow execution time fluctuates and it depends upon the type of Cloud resources used for executing the jobs. The workflow performance improved on better Cloud resources i.e. with more RAM and CPU. This verifies the claim made in this research study that the Cloud resource configuration should be known in order to reproduce the similar execution performance.

Once Cloud-aware provenance has been captured, it was analysed for correctness and completeness. This was discussed in Section 6.4.1 and 6.4.2. Table 6.2 showed that the correctness analysis approach performed fine and could detect any missing information in the collected Cloud-aware provenance. This correctness analysis was performed for each job. For complete workflow, the completeness analysis was performed. This analysis could identify all the jobs described in the workflow description and retrieve their Cloud-aware provenance and also workflow provenance. For

completeness check, it looked through all the jobs and checked whether they all had correct job-to-resource mapping information. The result in Table 6.4 shows that the completeness analysis was able to detect if there was missing job provenance. Since ReCAP depend on the captured Cloud-aware provenance for workflow reproducibility, it was important to verify if the collected provenance in the ReCAP was correct and complete. This also answers **Research Question 3** of this thesis.

The chapter then discussed the capability of the proposed approach in reproducing a workflow execution using the Cloud-aware provenance in Section 6.5. It is shown that the ReCAP approach is cable of capturing the Cloud-aware provenance of the previous workflow execution and then can use it to re-provision the similar resources on the Cloud in order to re-execute the workflow. In this experiment three different executed workflows were: 1) Montage, 2) ReconAll and 3) Wordcount. The result in Figure 6.7 showed that ReCAP can re-provision the similar execution resources on the Cloud. Figure 6.12 showed that the similar workflow execution performance was obtained by re-provisioning the similar execution infrastructure on the Cloud. Similarly, Tables 6.8 and 6.9 show that ReCAP was able to re-provision and re-execution the Montage workflow on the same resources on the Cloud which were used before. The ReconAll workflow was also executed using ReCAP. Since it is composed of only job, it uses only one virtual machine. Tables 6.10 and 6.11 show that ReCAP captured its Cloud-aware provenance and then was able to re-provision the same resource on the Cloud for its re-execution.

The workflow reproducibility is determined by comparing the provenance graphs. A detailed analysis of the provenance graph comparison is presented in Section 6.6. The Table 6.13 shows that the provenance structure comparison approaches perform as expected. The results show that the Numerical comparison approach provides more insight about the provenance. For instance, the Boolean comparison between workflows 317 and 207 failed. However, the Numerical analysis suggested that they were 96.51% similar (see Table 6.13). Further investigation into the graph showed that 317 was executed on the same infrastructure as was 207. However, job to host mapping differed because of the job scheduling of the workflow management system. This caused a change in the job to host edge in the provenance graph and consequently it affected the similarity result. From this analysis, it was concluded that the proposed provenance comparison algorithms performed according to their design. However, it was also found that job scheduling of WMS could affect the expected similarity outcome.

In Section 6.6.3, this chapter presented the result of workflow output comparison (discussed previously in Section 4.2.3). Figure 6.19 showed that the algorithm could iterate over the files produced by the workflow jobs on the Cloud and could compare them using their MD5 hash values. By comparing the MD5 values of all files produced by both the given workflows, the algorithm could provide a Boolean result. If all the files are same, the algorithm returns $True$ as a comparison result, otherwise it returns $False$. The results show that using MD5 for output comparison works consistently for deterministic workflows e.g. Montage. It was found that workflows i.e. Wordcount and Montage both produced the same output when they were reproduced. However, using MD5

in output comparison does not work in the case of ReconAll workflow. It was found that ReconAll produces a lot of files and some of them, mainly log files, include timestamps and machine name information. Due to the change in timestamps, the calculated hash values differ a great deal. However, it was found that the files related to thickness, area, and volume of the brain and the statistical data files were consistently identical in reproduced execution. From this analysis, it can be concluded that the proposed approach for workflow output comparison can work for multiple scientific domains especially with deterministic workflows and do not add machine or time specific information in the output.

By combining the outcomes from the provenance completeness analysis and provenance structure, infrastructure and output comparison, the workflow reproducibility was demonstrated. The Table 6.21 shows that ReCAP has successfully reproduced all three workflows. The ReconAll workflow was considered as reproduced because it uses the same infrastructure and it produced the same files related to thickness, area and volume of the brain. For verification purposes, a different provenance graph of Montage (wfID 535) was compared with a provenance graph of Montage (wfId 533). Although both workflows produced the same output, however ReCAP considered workflow 535 as 'not reproduced' because it was executed on different execution infrastructure. These results have shown that ReCAP can demonstrate workflow execution reproducibility by using the Cloud-aware provenance to re-provision similar resources on the Cloud. This answers **Research Question 4** of this thesis.

The chapter then analysed the performance and size overheads caused by the proposed mapping approaches. The size overhead caused by the proposed approach shows that 505.04 bytes (on average) are required to record job-to-Cloud resource mapping along with the CPU information. The size overhead increases to 925.14 bytes when the job file, which is stored on the Cloud, information is also recorded in the collected provenance. The results in Figure 6.14 show that the Static and Eager mapping approaches do not influence the workflow execution performance as these mapping approaches collect the Cloud-aware provenance information outside the virtual machine and job execution. The SNoHi mapping approach has a little impact on the workflow execution time as it collects machine information inside the job. The impact of this approach on the workflow execution time depends upon the time it takes to collect provenance data during the job execution. The overall analysis of provenance overhead suggests that the proposed mapping approaches do not affect a job execution because they operate from the outside of the virtual machine.

The next chapter, Chapter 7 concludes this research by providing the summary of this research work and by answering all the research questions individually. It also answers the research hypothesis in the light of this research work. It then discusses the key contributions made as a result of this research study and also presents a self-assessment by critically analysing the proposed work. This analysis is then linked with the future directions in which this research work can be taken.

Chapter 7

Conclusion and Future Directions

This chapter concludes the thesis and presents a summary of the findings of this research work. It presents the research questions (previously discussed in Section 1.4) and their answers, which have been resolved during the course of this research work. Based on their answers, it provides a resolution for the research hypothesis that under-pinned this entire research work. This chapter then highlights the key contributions made in this research work and discusses a few possible future directions for this research work.

7.1 Conclusions

This thesis has sought to test the following research hypothesis:

A Cloud-Aware provenance capturing approach can be used to collect provenance of the execution concerning Cloud-based scientific workflows to demonstrate the reproducibility of the workflow-based scientific experiments.

The aim of this research study has been to capture Cloud provenance information alongside workflow provenance data and to use the combined provenance for achieving strict workflow execution reproducibility by re-provisioning similar execution resources on a Cloud infrastructure. In order to answer the research hypothesis, an approach named ReCAP has been presented and discussed in Chapter 3 and 4. ReCAP has been designed in the light of the requirements identified in the workflow reproducibility framework for the Cloud (discussed in Section 3.3). ReCAP has also been modelled by analysing the real world user requirements of the N4U community (discussed in Section 3.5.1). ReCAP collected the workflow provenance from the underlying workflow management system and the Cloud provenance from the Cloud infrastructure, and then linked them together using the mapping algorithms discussed in Section 4.1. The mapping approaches mapped the workflow jobs onto the information about the Cloud resources, which were used to execute them.

This interlinked provenance information was named Cloud-aware provenance (CAP) in this thesis. This mapping information was then used to re-provision resources on the Cloud and to re-produce a workflow execution. Once an execution infrastructure was re-provisioned, the workflow was re-submitted for execution and its provenance was again captured. The provenance of the reproduced workflow was compared against the original workflow in order to determine the reproducibility.

The reproducibility of a workflow execution has been evaluated by comparing different aspects of the collected provenance graph and the workflow execution time. The results in Figure 6.11 and Figure 6.12 have shown that similar workflow execution performance can be achieved by re-executing the workflow on a similar execution infrastructure on the Cloud. Three different comparison operations, which include the 1) workflow graph structure comparison, 2) workflow outputs comparison and 2) the Cloud infrastructure comparison, were tested to verify workflow reproducibility on the Cloud. The graph structure comparison of provenance can detect any change either due a changed node or a changed edge in the given graphs (see results in Section 6.6.1). From infrastructure analysis, it was determined that ReCAP can re-provision similar resources on the Cloud using the Cloud-aware provenance. The workflow output comparison was applied on the workflow outputs to verify the reproduced outputs. From the results (see Tables 6.19 and 6.20), it was found that the workflow reproduced the same output and the comparison algorithm performed according to its design. By combining the outputs from these three comparisons, it was demonstrated (see Section 6.7) that the Cloud-aware provenance information could be used to demonstrate the reproducibility of workflow-based scientific experiments on the Cloud. In order to verify this, three different workflows 1) Montage from astronomy domain 2) ReconAll from neuroscience domain and 3) Wordcount - a simulated workflow for controlled analysis - were executed using ReCAP and their provenance was captured for analysis. The results were found to be consistent for all these three workflows. By using different workflows from different scientific domains, the applicability of the proposed approach ReCAP in different domains was also demonstrated.

In order to address the above hypothesis, a number of associated research questions (discussed in Section 1.4) were answered during the course of this work. The research questions along with their answers are given below.

1. **To what extent are the existing provenance capturing techniques adequate for capturing provenance of a Cloud-based scientific workflow?**

   In order to answer this question, an extensive literature review has been presented in Chapter 2. Different workflow management systems for the Grid and the Cloud, and their provenance capabilities have been analysed (see Section 2.5 and Section 2.6). The systems designed for the Grid can provide provenance information of a workflow execution, however they lack the support for the infrastructure information, which is required in the Cloud environment. Similarly, the approaches proposed in the Cloud environment provide different types of provenance data at different levels such as at the filesystem or the hypervisor levels. There are a few projects

such as Swift (Wilde et al. 2011) that provide support for workflow execution on the Cloud, however the collected provenance does not include Cloud resource information. There are a few approaches (Cruz et al. 2014; Groth et al. 2009) that consider the importance of Cloud resource information in the collected provenance. However, the collected information is not sufficient to achieve workflow execution reproducibility on the Cloud. Based on the literature analysis, it has been found that provenance can be collected at different levels i.e. at the workflow level, the hypervisor level and the filesystem level, however, the Cloud resource information is missing in them. Moreover, the Cloud infrastructure information is not mapped on to the workflow jobs, which was the main aim of this research study.

2. **How can provenance information be captured at different layers in a Cloud environment and interlinked?**

   In this research study, the application layer and infrastructure/virtualization layer are considered in a Cloud environment. The application layer is based on the workflow management system using the resources provided by the infrastructure layer. From the literature survey discussed in Chapter 2, it has been found that one can collect the workflow provenance and also the Cloud provenance at various levels. However, the interlink between a workflow job and the Cloud resource on which it was executed is missing. From a literature analysis, it has also been found that many of the existing workflow management systems already record some information about the machine on which a job was executed (see Table 2.2). Moreover, there are Cloud APIs that can interact with the Cloud middleware and can retrieve information about the running Cloud resources. Since many of the existing workflow management systems have the machine information, it can be used to establish a mapping between a workflow job and a detailed Cloud resource configuration. In this regard, different mapping mechanisms are required based on the used Cloud environment.

   There are two Cloud environments identified in Section 4.1 that drive the proposed job-to-Cloud resource mapping mechanisms. The first environment is named *Static* in which all the Cloud resources are provisioned before the workflow execution and they remain accessible on the Cloud long after the workflow finishes its execution. For such an environment, the Static mapping algorithm is proposed (discussed in Section 4.1.1). However, it was found (see Figure 6.9) that Pegasus takes around 25 seconds to register host information of a job in its provenance store. This delay may not cause issues in a Static environment, however in the *Dynamic* Cloud environment, a resource will no longer be accessible during this delayed period. Therefore, the Static mapping algorithm will not work for the *Dynamic* Cloud environment, in which resources are released once a job has completed.

   For the *Dynamic* environment, two mapping algorithms i.e. the Eager (discussed in Section 4.1.2) and Lazy (discussed in Section 4.1.3) mappings have been proposed. The Eager approach attempts to establish a temporary mapping between a job and a Cloud resource as early as possible by using the capabilities of the underlying execution framework. Since this pro-

totype was designed and implemented using Pegasus, which works with the Condor cluster, the implemented Eager approach is Condor dependent. To overcome this dependency, the Lazy approach has been devised which periodically monitors the Cloud infrastructure and creates a temporary record of the Cloud infrastructure. This record is then compared with the workflow provenance and the job-to-Cloud resource mapping is deduced once a workflow has completed.

This Lazy approach relies on the fact that a workflow management system will eventually record the host information for a job in its provenance store. However, there could be a scenario in which the workflow management system does not support the host information in its collected provenance information or a scenario in which the Grid-based execution infrastructure is used (which is the case for N4U). Consequently the SNoHi mapping (discussed in Section 4.1.4) approach has been presented. In this approach, the job is augmented with the capability to record execution machine information as XML (see Appendix B.1). This XML is then stored in the provenance store. For a Grid-based environment such as that used in N4U, this XML can provide the machine information including the machine name, IP, CPUs, RAM, Hard disk and operating system. For the Cloud-based environment, this XML can then be used, along with the Static or Lazy approaches (depending upon the Cloud environment), to establish the final job-to-Cloud resource mapping.

Based on this discussion, it can be concluded that the proposed mapping approaches can be used to capture Cloud-aware provenance in different Cloud environments i.e. Static or Dynamic. The Static mapping approach can aid workflow management systems such as Pegasus that work in a Static Cloud environment. It can also be integrated with CRISTAL as used in the N4U project but N4U uses a Grid infrastructure. For the dynamic Cloud environment, the Eager and Lazy approaches can also be integrated with Pegasus provided it can provision resources dynamically. In order to facilitate systems such as Chimera that do not maintain host information, the SNoHi approach can be used. The overhead analysis results (see Figure 6.14) show that the proposed approaches do not cause any performance overhead, unlike some existing systems such as Matriohska (Cruz et al. 2014), because the proposed approaches work outside the virtual machines.

3. **How can correctness and completeness of the collected provenance be measured?**

   Section 3.4 discussed the correctness and completeness of the collected provenance information in the light of the literature review. The correctness was defined as "the extent to which provenance data are correct and free of error." This definition has been redefined as *"the extent to which the Cloud specific information is accurately acquired in Cloud-aware provenance"* in the context of this research study. Since this work focuses on the Cloud-aware provenance, the correctness of the provenance in this work is measured by determining the presence of the identified Cloud specific parameters such as the RAM, CPU, Hard Disk, Flavour and Image name in the provenance. In this thesis, the correctness is shown in two ways i.e., (1) through

analysing the provenance and detecting the presence of the Cloud resource configurations and (2) by evaluating the impact of collected parameters on an individual job and workflow. The result presented in Table 6.2 shows the application of the correctness approach (discussed in Section 4.2.2.1) on the captured Cloud-aware provenance. The devised approach looks for the identified Cloud parameters in the Cloud-aware provenance and raises the $False$ flag if any of the identified parameter is missing. With this approach, a user or system, ReCAP in this case, can verify the collected Cloud-aware provenance and identify if there is missing information. Since ReCAP relies upon this information in re-provisioning resources on the Cloud, it is essential to ensure that the collected provenance does not have any missing information. Furthermore, the correctness of the collected provenance has also been verified by confirming the significance of the identified parameters on the job's failure rate and the job execution performance. The results in Figures 6.1 and 6.2 confirmed that these parameters could impact the job's performance. Moreover, the impact of various resource configurations on the workflow execution was evaluated (see Figure 6.6) and it was found that the workflow execution time was dependent upon the underlying resource configurations.

The completeness of the collected provenance is determined by analysing the workflow structure and its collected Cloud-aware provenance. In this analysis, the Cloud-aware provenance for all the jobs defined in the workflow description are analysed. If there is any job missing in the mapping or there is any Cloud resource configuration missing, this means that the collected provenance is not complete. This type of information is essential in case of ReCAP because it relies upon the collected Cloud-aware provenance information. If the collected provenance is not complete, this will affect ReCAP's ability to re-provision the same resources on the Cloud for workflow re-execution. Moreover, this will also affect the comparison algorithms because they include infrastructure information in the graph.

4. **How can we demonstrate workflow reproducibility on the Cloud infrastructure using Cloud-aware provenance?**

   The proposed approach in this thesis is capable of reproducing a workflow execution on the similar Cloud infrastructure using Cloud-aware provenance. In order to achieve this, the prototype has designed and implemented the WF-Repeat component (discussed in 4.3) that takes a workflow and its associated Cloud-aware provenance, and re-executes it on the Cloud. Since ReCAP stores the workflow description and its associated configuration files in its database, the WF-Repeat component can retrieve the original workflow from the database. In addition, since ReCAP also has the job-to-Cloud resource mapping of the workflow, it can re-provision those resources on the Cloud by creating resource requests using the vCPUs, RAM, Hard Disk and OS information. The results shown in Figures 6.6 and 6.7 confirmed that the proposed approach can re-provision the resources on the Cloud with the same configurations and can re-execute the workflow. Figures 6.11 and 6.12 showed that the workflow execution time remained almost similar on the re-provisioned resources.

From the results (shown in Table 6.13), it has been demonstrated that the proposed provenance structure comparison algorithms perform according to the design (discussed in Section 4.2.1). The proposed provenance graph comparison algorithms include infrastructure information in the provenance graph as well. They can detect a slight change in the graph due to either a change in the workflow structure or the infrastructure information added into the graph. From the combined comparison of workflow structure and Cloud infrastructure, it was verified whether the workflow execution was reproduced on the same infrastructure on the Cloud that was used in earlier execution. While comparing provenance graphs, it was found that two given provenance graphs could exhibit some differences despite being executed on same infrastructure on the Cloud, therefore they produced < 100% similarity score (see Table 6.13). With deeper analysis using the detailed infrastructure comparison i.e. using job-to-resource mapping (shown in Tables 6.17 and 6.18)), it was found that this was due to the job scheduling, which was outside the scope of this thesis, that caused changes in job to VM edges in the provenance graph. This is why it is suggested that Cloud-aware provenance should be incorporated in the job scheduling algorithms (discussed in Section 7.5).

Since workflow reproducibility is nothing without producing the same results, the workflow outputs were also compared using the workflow output comparison approach presented in Algorithm 4.2.3. From the results (see Tables 6.19, 6.20), it has been found that the algorithm can compare the outputs produced by the workflows and can determine whether the workflow outputs are reproduced. The use of the MD5 hashing approach in this algorithm proves fruitful in cases where workflows are deterministic such as Montage or Wordcount. Here deterministic means that these workflows will always produce the same output given the inputs are the same. However, this approach was not useful for cases in which the produced output is dependent upon a random seed value, or timestamps. This was the case found when analysing the outputs of the ReconAll workflow. The MD5 hash of some of its outputs i.e. mainly log files (snippets shown in Appendix B.2) could not be matched in the reproduced execution because these files contain different timestamps. Excluding these files, it was found that the remaining files dealing with actual thickness, area and volume and statistical analysis of the given neuro-image were identical in the reproduced execution (see Table B.1). In order to mitigate this, it is suggested that the workflow or application developer should provide an output comparison utility that can be used for output comparison. Overall the proposed output comparison algorithm is consistent across different workflows from different domains. Moreover, the output comparison algorithm will perform better if the deterministic files are known.

By combining the outputs produced by structural, infrastructure and output comparisons, the workflow execution reproducibility has been demonstrated (see Table 6.7). It has also been demonstrated that ReCAP can capture the Cloud-aware provenance and can verify its correctness and completeness (see Tables 6.3 and 6.5). It then can use this information to reproduce a workflow execution by re-provisioning similar execution infrastructure on the Cloud.

7.2 Key Contributions

The following are the key contributions of this research study.

- **Mapping Workflow Provenance with Cloud infrastructure information**

  As the Cloud is dynamic and different from other distributed environments such as the Grid, a workflow reproducibility system, ReCAP, has been proposed (discussed in Section 3.3) that keeps Cloud infrastructure at its centre. Based on this framework, an automated mapping approach that works outside the virtual machines has been devised to retrieve the configuration of Cloud resources which were used to execute the jobs of a workflow. In this approach, no additional jobs or processes need to be running within a virtual machine. Two usecases of the Cloud environments i.e. the Static and the Dynamic have been identified (see Section 4.1) and different mapping mechanisms have been designed to mitigate the challenges of these use cases. These approaches are given below.

  (a) **Mapping approaches for Static Environment:** For the Static environment, the Static mapping approach has been discussed in Section 4.1.1 and evaluated in Chapter 6. This approach can work in situations in which a resource remains on the Cloud after a job finishes its execution. However, this may not be the case in the Dynamic Cloud scenario in which a resource will no longer exist once the job finishes its execution. To mitigate the dynamic scenario, the dynamic mapping approaches i.e. a) Eager and b) Lazy have been proposed.

  (b) **Mapping approaches for Dynamic Environment:** As the Static approach relies on the host information for a job available in the workflow management system database, it was found (see Figures 6.8 and 6.9) that Pegasus took around 25 seconds to register this information in its database. This is a significant time delay and there is no guarantee that the Cloud resource would remain accessible on the Cloud for this amount of time in the Dynamic scenario. This is why two other mapping approaches i.e. the Eager approach (discussed in Section 4.1.2) and the Lazy approach (discussed in Section 4.1.3) have been proposed for the Dynamic environment. The result in Figure 6.10 showed that the Eager approach could monitor and capture the job-to-Cloud resource mapping much earlier than the Static approach that relied upon the information from the WMS database, which can take time.

- **Workflow Provenance Comparison**

  In this thesis, ReCAP collects Cloud-aware provenance and uses it in the re-execution of a workflow. As discussed in Section 3.3, the collected provenance is analysed on different levels i.e. provenance structural comparison, infrastructure comparison and output comparison to

verify the workflow reproducibility. In this regard, different approaches have been devised and presented in this thesis. These approaches are given below.

(a) **Provenance Comparison Algorithms:** In this research, the collected workflow provenance has been converted into a graph. This graph also includes the Cloud resource information. The algorithm compares the graph of the original execution with the reproduced workflow provenance and determines the extent to which the execution traces are similar. The algorithm also takes various factors into account, which have been discussed in Section 4.2. One of these factors is the Cloud infrastructure information in the comparison, which was missing in earlier approaches (e.g. Missier et al. 2013). In order to perform the comparison on two given provenance graphs, two algorithms (discussed in Section 4.2) have been devised in this thesis. One version of this comparison provides the result as True or False. For it, the given provenance graphs are either the same or different. In order to quantitatively measure the similarity between two provenance graphs, a numerical comparison approach is also presented in Section 4.2.1.3. The results discussed in Section 6.6.1 have verified that both these algorithms performed according to their design. They can detect a single change in the given provenance graphs. However, it is possible that two workflows were executed on the same infrastructure yet they produced provenance graphs that are not similar (discussed in Section 6.6.1). With further analysis, it was found that it was due to the job scheduling that introduced a change in the edges connecting certain jobs to VMs in the graphs. The results in Tables 6.17 and 6.18 have confirmed this scenario. Overall, the results showed that the proposed provenance comparison approaches can compare the provenance graphs and can measure the structural similarity between them.

(b) **Workflow Output Comparison:** There is also a workflow output comparison algorithm presented in Section 4.2.3 that aids in comparing the outputs produced by the workflows. This algorithm uses a hashing algorithm while comparing the workflow outputs. The results in Section 6.6.3 have shown that this approach is feasible in cases where workflows such as Montage are deterministic. During the analysis of the ReconAll workflow, it was found (see Section 6.6.3) that the hashing approach applied for workflow output comparison did not work for its output files. Further analysis of the outputs showed that it was due to the introduction of timestamps in the generated output logs (shown in Listings B.6 and B.7) that caused the hashing output comparison ineffective. However, all the other files related to brain thickness, area or volume produced the same hash values (see Table B.1) and the proposed output comparison algorithm worked fine for these files. Based on this particular analysis, it was also suggested that the workflow or application developers should also provide a mechanism or tool to compare the outputs produced by their workflows. Such tools can then be integrated in the workflow output comparison. Overall, the analysis of the workflow output comparison (shown in Tables 6.19, 6.20 and B.1) has shown that the proposed approach is workable for scientific

workflows, especially the deterministic workflows such as Montage.

(c) **Provenance Correctness and Completeness Analysis:** With reference to the provenance correctness and completeness discussed in Section 3.4, this thesis also presents two approaches for analysing the provenance correctness and completeness of the collected Cloud-aware provenance. These approaches have been discussed in Sections 4.2.2.1 and 4.2.2.2 respectively. The purpose of determining the correctness and completeness of the collected provenance is to ensure that the captured provenance is correct and reliable to be used for workflow re-execution. The results in Tables 6.3 and 6.5 have shown that these approaches can detect missing parameters in the collected Cloud-aware provenance information. This analysis facilitates trust in the captured provenance by ensuring ReCAP has collected the Cloud infrastructure information correctly for all the jobs in the workflow.

- **Workflow Execution Reproducibility on the Cloud**

  In this research work, it has been demonstrated, using the ReCAP approach, that workflow execution reproducibility can be achieved on the Cloud by re-provisioning similar execution resources on the Cloud. In order to achieve this, ReCAP captures Cloud-aware provenance and uses it to re-provision resources on the Cloud. The experimental result (in Figure 6.6) showed that the types of Cloud resource configuration used can impact the workflow execution. This result confirmed that the resource configuration information should be known in order to reproduce a workflow execution with a similar performance on the Cloud.

  In ReCAP, a component, WF-Repeat, has been designed and implemented that re-provisions resources on the Cloud and re-executes the workflow on them. The result shown in Figure 6.7 illustrates that the system can successfully re-provision the same resources on the Cloud. The results in Figures 6.11 and 6.12 also confirm that a similar execution performance has been achieved by using the Cloud-aware provenance to re-provision the Cloud resources. This is particularly important for workflows such as the Epigenome workflow from the genomic experiment (Pietri et al. 2014).

7.3 Applicability of the Research

The following are potential areas in which this research work can also be applied and used.

- **Experiment Reproducibility in Neuroscience**

  The workflows (also known as pipelines) used in Neuroimaging analysis can produce different results depending upon the computing platform where they were executed (Gronenschild et al. 2012). Kanwal et al. (2015) also highlighted the significance of reproducibility to verify the result outcomes of complex workflows used in modern experiments. N4U is one such

project that provides analysis tools such as CRISTAL for performing user analysis on the neuro-images. It was part of the user requirements (see Section 3.5.1) of N4U to devise a mechanism that can assist in validating the workflow results using provenance. The proposed system, ReCAP, can facilitate these domains in demonstrating the workflow execution reproducibility on a Cloud infrastructure, where resources are dynamic and provisioned on-demand. Since ReCAP can capture details about the execution infrastructure, it can use this information to re-provision similar execution infrastructures for workflow re-execution. In this manner, the user will get the same infrastructure with the same hardware and software configuration, which is suggested by (Gronenschild et al. 2012; Glatard et al. 2015). It has been demonstrated using different workflows such as Montage from astronomy and ReconAll from N4U that ReCAP can aid in reproducing a workflow execution on the Cloud by capturing Cloud-aware provenance and then using it to re-provision same Cloud infrastructure for workflow re-execution. Moreover, ReCAP also proposed a workflow output comparison approach using the MD5 hashing mechanism. It has been applied on the outputs produced by the ReconAll workflow. By comparing the files related to brain area, thickness and volume and files with statistical data about the brain analysis, this approach can validate a workflow re-execution.

- **Reproducible Research**

  Much work (e.g. Bechhofer et al. 2010; Gandrud 2013) is being carried out in the domain of reproducible research in which efforts have been made to devise approaches that can help in reproducing the results claimed in the research papers. The motivation for this research originated by looking at the statistical data suggesting that the results in many papers were not reproducible due to several reasons. Since this research study aims at workflow reproducibility on the Cloud by providing Cloud-aware provenance information, it could be used to verify and achieve reproducible research for the workflow-based experiments on the Cloud. Since ReCAP can provide infrastructure information used by a workflow, this information can be reused to re-execute workflow execution on similar infrastructures. It has been demonstrated (see Section 6.5) that the resource configuration on a Cloud can affect on workflow execution performance. With the help of Cloud-aware provenance produced by ReCAP, a researcher can share the execution infrastructure details with the colleagues who can then use this information to re-execute the experiment on similar infrastructures on the Cloud.

- **Extensible Software Design**

  Since the proposed design is plugin-based, it can accommodate or be integrated with other systems. For example, Kepler also provides a plugin approach but it lacks an automatic capturing mechanism in its provenance model to collect the information about the Cloud resources used for a workflow execution. This proposed solution can be made part of the Kepler provenance framework by implementing the Kepler specific plugins in ReCAP and using Kepler as the workflow management system. Similarly, ReCAP can be integrated with Taverna as there

is an on-going project in Taverna to include information about the execution environment in its workflow provenance using the Research Object Bundle[1]. It can also be integrated with CRISTAL, which provides support for workflow evolution and detailed provenance about each user interaction defined inside CRISTAL. By integrating ReCAP with CRISTAL, it will also be able to track the Cloud infrastructure configurations used during a workflow execution.

7.4 Critical Analysis of ReCAP

The proposed system, ReCAP, has been presented to capture Cloud-aware provenance and to reproduce workflow execution in a Cloud infrastructure. In order to achieve the workflow reproducibility, a set of requirements has been identified in Section 3.3. Out of these requirements, the ReCAP database schema does not provide support for workflow versioning because the focus of this work was on the workflow execution on the Cloud. Although it stores the original workflow and its associated files, the track of the evolution of the workflow is missing. Similarly, ReCAP re-creates a virtual machine by using the same image that was used in the previous virtual resource. It was done to provide the same software stack i.e. operating system and the software libraries etc., for the job. This means that ReCAP does not provide a configuration mechanism to install and configure all the required software on the newly provisioned virtual machine. This can be resolved by using an automation tool such as Chef or Puppet. These tools enable a user to configure the required software on the newly acquired virtual machine. This work is highlighted as potential future work.

In order to re-provision resources for the workflow jobs, ReCAP relies upon the Cloud-aware provenance information that is captured using the proposed mapping approaches (as discussed in Section 4.1). At present, the WF-Repeat component sends resource provisioning requests for all the required Cloud resources for a workflow at one time. This simple mechanism may face a couple of challenges. One of the challenges is related to the user quota. Some Cloud Providers such as Open Science Data Cloud (OSDC) assigns a resource quota to each user. This governs the number of resources users can have on the Cloud. Therefore, the first challenge the WF-Repeat component may encounter is over-provisioning of the resources. The other challenge caused by the simple provisioning mechanism may be the in-efficient use of the resources. In order to understand this, a workflow example (shown in Figure 7.1) with six jobs is given. This workflow used six unique virtual machines during its entire execution. This means that the workflow jobs were scheduled to six different virtual machines.

The analysis of the graph structure of this workflow shows that a maximum of two jobs can run in parallel at any given time after the first job A. Since the WF-Repeat component is not a fully featured workflow management system and does not provision resources based on the workflow

[1] https://researchobject.github.io/specifications/bundle/

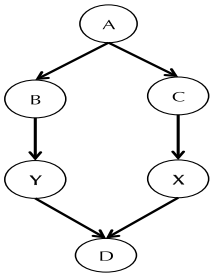


Figure 7.1: A workflow example with six jobs

graph structure, this is possible that it re-provisions all six machines at once. Since a maximum of two jobs can run in parallel in the given workflow example, this means that only two virtual machines will be in use for job execution and the remaining machines will remain idle. This leads to in-efficient resource utilization.

At the time of writing ReCAP has been tested on a single Cloud infrastructure offered by the OSDC. It has not been tested in a scenario in which the execution infrastructure is established by provisioning resources on multiple Cloud Providers. For instance, it is possible to create a virtual cluster on a set of machines provisioned on the Amazon EC2 and the Windows Azure Cloud[2]. The mapping approaches in ReCAP are only able to map the workflow job to the Cloud resources accessible to it. In order to access multiple Cloud Providers, it will require access to the user credentials for each Cloud Provider will be required. This feature is missing at the moment and could be one source of future work. Furthermore, ReCAP has been evaluated with different workflows with varying number of jobs, which were executed over a small number of Cloud resources. However, it is not tested over a large pool of Cloud resources. This was mainly due to the limited resources available during the testing phase. Its scalability evaluation over a large pool of Cloud resources can be another potential future work.

7.5 Future Directions

A few of the possible future directions are discussed below.

- **Integration with WMS**

  In the implemented prototype, ReCAP has been integrated and tested with Pegasus only. Although one of its concepts is used in CRISTAL job submission to retrieve provenance information about an execution machine, its integration with other WMS has not been tested. Nonetheless, since the proposed design is plugin-based, it can accommodate or be integrated with other systems. For example, Kepler or VisTrails also provides a plugin approach but

[2]http://azure.microsoft.com

it lacks a detailed Cloud resource information in the collected workflow provenance. The proposed solution from this work can be made part of Kepler's provenance framework by implementing Kepler specific plugins in ReCAP and using Kepler as the workflow management system. This will test the potential of ReCAP and its plugin-based software architecture with other WMSs. Similarly, a plugin can be developed to integrate ReCAP with CRISTAL that has been used in the N4U project.

- **Integration with Version Control**

  As discussed in the Workflow reproducibility model, keeping track of workflow versions and their associated files has been highlighted in the literature as an important point for workflow reproducibility. At present, the ReCAP's database schema does not support multiple versions of a workflow and their evolution. However, this can be achieved by integrating ReCAP with systems like CRISTAL or VisTrails that can keep track of such evolutions. It is also possible to integrate ReCAP with version control system such as GitHub[3] or SVN[4] as it has been done in the literature with other approaches.

- **Integration with the Infrastructure Automation Tools**

  At present, ReCAP relies upon the OS image stored on the Cloud as the basis for providing all the required software and the operating system stack for the virtual resource. However, this limitation can be avoided with a mechanism that can configure a base virtual machine to the required level by installing and configuring all the required software. One such approach (Klinginsmith, Mahoui and Wu 2011) helps in re-installing and reconfiguring a software environment on top of VMs. The work proposed in this thesis can be integrated with such approaches to provide a fully automated mechanism to re-provision similar resources with a completely configurable software stack on the Cloud for workflow re-execution. With this combined approach, the limitation of using pre-configured VM can be avoided.

- **Integrating CAP with Job Scheduling Algorithm**

  Besides these, the Cloud-aware provenance information collected by ReCAP can also be used in workflow planning and scheduling algorithms. This enables them to make efficient use of the Cloud resources by intelligently provisioning execution resources on-demand and scheduling jobs onto them. Such a job scheduling algorithm will make sure that a workflow job will execute on the same resource as was the case in its previous execution.

- **Quota Driven Resource Provisioning**

  In the Cloud-based Workflow Manager (presented and discussed previously in Section 4.4), the VMs are provisioned based on the ready jobs of the given workflow. The resource provisioning mechanism does not incorporate the user allowed quota on the Cloud, which is the

[3]https://github.com/
[4]https://subversion.apache.org/

case in the real world. This means that the resource provisioning request will fail if the user has already completed his allowed quota. To overcome this, an efficient resource provisioning is required as part of the workflow engine.

- **In Multi-Cloud Environment**

  So far, this research work has concentrated on collecting Cloud resource information from a single Cloud provider i.e. one Cloud middleware. However, it is possible that a workflow is executed over resources provisioned in multiple Cloud infrastructures offered by multiple Cloud providers. For instance, a workflow can be executed on resources acquired from the Amazon EC2[5] and Microsoft Azure Cloud infrastructures. A future study could be made in this direction to collect and map the Cloud resource information, which may be collected from multiple Cloud providers, to the workflow jobs. It is also possible that different Cloud Providers offer different resource configurations. For instance, Amazon EC2 offers an exhaustive list of resource types, many of which are different from the default resource configurations provided by the OSDC and OpenStack. Therefore, it is possible that such resource types are not available in other Cloud providers, thus re-provisioning similar resources becomes a challenge. Moreover, it also requires extending ReCAP with more configurations to support multiple user credentials for accessing multiple Cloud providers and more general interfaces to interact with them.

- **Weighted Provenance Graph Comparison**

  In the proposed provenance graph comparison approach, all nodes and edges are given equal importance. In future, different graph edges can be prioritized by assigning weights to them depending upon their critical nature in the workflow structure. This can be achieved by incorporating semantic information such as the one suggested in (Alper et al. 2013), which uses *motifs* to provide conceptual abstractions for the workflow jobs and to categories the workflow jobs. Similarly, in the proposed model different aspects of the workflow reproducibility such as workflow structure, Cloud resource information and the produced outputs are also treated separately. As a potential future work, this research intends to explore the feasibility of combining aforementioned parameters into a single integrated comparison approach.

In conclusion this thesis work has attempted to capture Cloud provenance and to link it with the workflow provenance of a workflow executing on a Cloud infrastructure. In this regard, the ReCAP framework has been proposed that can capture and establish mapping between a workflow job and the Cloud resources used for execution. The proposed mapping approaches work outside the virtual machine; therefore, they do not affect a job execution. This produces Cloud-aware provenance that helps in re-provisioning Cloud resources with same configurations to re-execute a workflow. In this way, a researcher can recreate an execution infrastructure of the same configurations as were

[5]http://aws.amazone.com/ec2

used in earlier executions, for workflow execution reproducibility. This thesis has demonstrated that ReCAP can capture and re-provision Cloud resources with the same configuration for workflow re-execution. It has also demonstrated that Cloud resource configurations can affect a job performance that consequently affects workflow performance. Different workflows from different scientific domains, such as astronomy and neuroscience, of varying number of jobs have been used to evaluate this work. This shows that the proposed approach can potentially be applied to multiple domains especially in neuroscience and in reproducible research. Further work is required to enable this approach to work in multi-cloud environments. Moreover, the use of Cloud-aware provenance in job scheduling and resource provisioning algorithms would enable efficient use of resources for job execution on the Cloud for reproducibility purposes.

References

Abbadi, Imam M. and John Lyle (2011). "Challenges for Provenance in Cloud Computing". In: *3rd USENIX Workshop on the Theory and Practice of Provenance* 3 (cit. on pp. 21, 30).

Abramovici, Alex et al. (1992). "LIGO: The laser interferometer gravitational-wave observatory". In: *Science* 256.5055, pp. 325–333 (cit. on p. 1).

Abrishami, Saeid, Mahmoud Naghibzadeh and Dick H.J. Epema (2013). "Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds". In: *Future Generation Computer Systems* 29.1. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures, pp. 158 –169. URL: http://www.sciencedirect.com/science/article/pii/S0167739X12001008 (cit. on p. 50).

Alper, P. et al. (2013). "Small Is Beautiful: Summarizing Scientific Workflows Using Semantic Annotations". In: *Big Data (BigData Congress), 2013 IEEE International Congress on*, pp. 318–325. DOI: 10.1109/BigData.Congress.2013.49 (cit. on p. 163).

Altintas, Ilkay, Oscar Barney and Efrat Jaeger-Frank (2006). "Provenance collection support in the kepler scientific workflow system". In: *Provenance and annotation of data*. Springer, pp. 118–132 (cit. on p. 28).

Azarnoosh, S. et al. (2013). "Introducing PRECIP: An API for Managing Repeatable Experiments in the Cloud". In: *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*. Vol. 2, pp. 19–26 (cit. on pp. 2, 40).

Banati, A., P. Kacsuk and M. Kozlovszky (2015). "Four level provenance support to achieve portable reproducibility of scientific workflows". In: *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on*, pp. 241–244. DOI: 10.1109/MIPRO.2015.7160272 (cit. on p. 47).

Barga, Roger S. and Luciano A. Digiampietri (2008). "Automatic Capture and Efficient Storage of e-Science Experiment Provenance". In: *Concurr. Comput. : Pract. Exper.* 20.5 (Apr. 2008), pp. 419–429. URL: http://dx.doi.org/10.1002/cpe.v20:5 (cit. on p. 24).

Barga, Roger S et al. (2010). "Provenance for Scientific Workflows Towards Reproducible Research". In: *Data Eng.* Bull. 33.3, pp. 50–58 (cit. on p. 49).

Bechhofer, Sean et al. (2010). "Research Objects: Towards Exchange and Reuse of Digital Knowledge". In: *The Future of the Web for Collaborative Science*. DOI: http://dx.doi.org/10.1038/npre.2010.4626.1 (cit. on p. 159).

Belhajjame, Khalid et al. (2012). "Why Workflows Break - Understanding and Combating Decay in Taverna Workflows". In: *Proceedings of the 2012 IEEE 8th International Conference on E-Science (e-Science)*. E-SCIENCE '12. Washington, DC, USA: IEEE Computer Society, pp. 1–9. URL: http://dx.doi.org/10.1109/eScience.2012.6404482 (cit. on pp. a, 2, 4).

Boettiger, Carl (2015). "An Introduction to Docker for Reproducible Research". In: *SIGOPS Oper. Syst. Rev.* 49.1 (Jan. 2015), pp. 71–79. URL: http://doi.acm.org/10.1145/2723872.2723882 (cit. on p. 37).

Bose, Rajendra and James Frew (2005). "Lineage Retrieval for Scientific Data Processing: A Survey". In: *ACM Comput. Surv.* 37.1 (Mar. 2005), pp. 1–28. URL: http://doi.acm.org/10.1145/1057977.1057978 (cit. on p. 22).

Branson, Andrew et al. (2012). "Workflow Orchestration and Provenance Management with CRISTAL". In: *Future Generation Computer Systems* (cit. on pp. 3, 5, 19, 26, 49).

C., Stodden Victoria (2010). "Reproducible Research: Addressing the Need for Data and Code Sharing in Computational Science". In: *Computing in Science & Engineering* 12. DOI: http://dx.doi.org/10.1109/MCSE.2010.113 (cit. on pp. 35, 46, 47).

CERNVM (2015). *CERNVM Software Appliance*. http://cernvm.cern.ch/. Online; Last Visited on 12-Sept-2015 (cit. on pp. 19, 57).

Chapman, Adriane P., H. V. Jagadish and Prakash Ramanan (2008). "Efficient Provenance Storage". In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. Vancouver, Canada: ACM, pp. 993–1006. URL: http://doi.acm.org/10.1145/1376616.1376715 (cit. on p. 24).

Chatrchyan, S et al. (2008). "The CMS experiment at the CERN LHC". In: *Journal of Instrumentation* 3.08, S08004 (cit. on p. 15).

Cheah, You-Wei and B. Plale (2012). "Provenance analysis: Towards quality provenance". In: *E-Science (e-Science), 2012 IEEE 8th International Conference on*, pp. 1–8. DOI: 10.1109/eScience.2012.6404480 (cit. on pp. 50–52, 90).

Chef. *Chef*. http://www.opscode.com/chef/. Online; Last Visited on 12-Sept-2015 (cit. on p. 39).

Chen, Weiwei and Ewa Deelman (2011). "Workflow Overhead Analysis and Optimizations". In: *Proceedings of the 6th Workshop on Workflows in Support of Large-scale Science*. WORKS '11. New York, NY, USA: ACM, pp. 11–20. URL: http://doi.acm.org/10.1145/2110497.2110500 (cit. on pp. 96, 110).

Chen, Weiwei and Ewa Deelman (2012). "Workflowsim: A toolkit for simulating scientific workflows in distributed environments". In: *E-Science (e-Science), 2012 IEEE 8th International Conference on*. IEEE, pp. 1–8 (cit. on pp. 108, 109).

Chirigati, Fernando, Dennis Shasha and Juliana Freire (2013). "ReproZip: Using Provenance to Support Computational Reproducibility". In: *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance*. TaPP '13. Lombard, Illinois: USENIX Association, 1:1–1:4. URL: http://dl.acm.org/citation.cfm?id=2482949.2482951 (cit. on pp. 36, 37).

Cruz, Sérgio Manuel Serra da et al. (2008). "Provenance Services for Distributed Workflows". In: *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 526–533. DOI: http://dx.doi.org/10.1109/CCGRID.2008.73 (cit. on p. 32).

Cruz, Sergio Manuel Serra da et al. (2014). "Collecting Cloud Provenance Metadata with Matriohska: A Case Study with Genomic Workflows". In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. SAC '14. Gyeongju, Republic of Korea: ACM, pp. 351–356. URL: http://doi.acm.org/10.1145/2554850.2555066 (cit. on pp. 32, 152, 153).

Cruz, S.M.S. da, M.L.M. Campos and M. Mattoso (2009). "Towards a Taxonomy of Provenance in Scientific Workflow Management Systems". In: *Services - I, 2009 World Conference on*, pp. 259–266. DOI: 10.1109/SERVICES-I.2009.18 (cit. on p. 23).

DAGMan (2014). http://research.cs.wisc.edu/htcondor/dagman/dagman.htm. Online; Last Visited on 12-Sept-2015 (cit. on p. 27).

Dai, Chenyun et al. (2008). "An Approach to Evaluate Data Trustworthiness Based on Data Provenance". In: *Proceedings of the 5th VLDB Workshop on Secure Data Management*. SDM '08. Auckland, New Zealand: Springer-Verlag, pp. 82–98. URL: http://dx.doi.org/10.1007/978-3-540-85259-9_6 (cit. on p. 50).

Davidson, Susan B. and Juliana Freire (2008). "Provenance and Scientific Workflows: Challenges and Opportunities". In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. Vancouver, Canada: ACM, pp. 1345–1350. URL: http://doi.acm.org/10.1145/1376616.1376772 (cit. on p. 16).

Deelman, E. and Y. Gil (2006). "Managing Large-Scale Scientific Workflows in Distributed Environments: Experiences and Challenges". In: *e-Science and Grid Computing, 2006. e-Science '06. Second IEEE International Conference on*, pp. 144–144. DOI: 10.1109/E-SCIENCE.2006.261077 (cit. on p. 16).

Deelman, Ewa et al. (2004). "Pegasus: Mapping Scientific Workflows onto the Grid". English. In: *Grid Computing*. Ed. by MariosD. Dikaiakos. Vol. 3165. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 11–20. URL: http://dx.doi.org/10.1007/978-3-540-28642-4_2 (cit. on pp. 5, 27).

Deelman, Ewa et al. (2005). "Pegasus: a framework for mapping complex scientific workflows onto distributed systems". In: *SCIENTIFIC PROGRAMMING JOURNAL* 13, pp. 219–237 (cit. on pp. 27, 56).

Deelman, Ewa et al. (2008a). "The Cost of Doing Science on the Cloud: The Montage Example". In: *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. SC '08. Austin, Texas: IEEE Press, 50:1–50:12. URL: http://dl.acm.org/citation.cfm?id=1413370.1413421 (cit. on pp. 50, 105, 106).

Deelman, Ewa et al. (2008b). "Workflows and e-Science: An overview of workflow system features and capabilities". In: (cit. on p. 2).

Deelman, Ewa et al. (2009). "Workflows and e-Science: An overview of workflow system features and capabilities". In: *Future Generation Computer Systems* 25.5, pp. 528 –540. URL: http://www.sciencedirect.com/science/article/pii/S0167739X08000861 (cit. on p. 16).

Fahringer, T., Jun Qin and S. Hainzer (2005). "Specification of grid workflow applications with AGWL: an Abstract Grid Workflow Language". In: *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*. Vol. 2, 676–685 Vol. 2. DOI: 10.1109/CCGRID.2005.1558629 (cit. on p. 16).

Foster, I. et al. (2002). "Chimera: a virtual data system for representing, querying, and automating data derivation". In: *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*, pp. 37–46 (cit. on pp. 24, 81).

Foster, I. et al. (2008). "Cloud Computing and Grid Computing 360-Degree Compared". In: *Grid Computing Environments Workshop, 2008. GCE '08*, pp. 1–10 (cit. on pp. 3, 20, 21, 28).

Foster, Ian and Carl Kesselman, eds. (1999). *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. (cit. on p. 1).

Freire, Juliana et al. (2008). "Provenance for Computational Tasks: A Survey". In: *Computing in Science and Engg.* 10.3 (May 2008), pp. 11–21. URL: http://dx.doi.org/10.1109/MCSE.2008.79 (cit. on pp. 2, 17, 45).

Freire, Juliana et al. (2014). "Reproducibility using vistrails". In: *Implementing Reproducible Research*, p. 33 (cit. on p. 17).

Gadelha Jr., Luiz M. R. et al. (2013). "Provenance Traces of the Swift Parallel Scripting System". In: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. EDBT '13. Genoa, Italy: ACM, pp. 325–326. URL: http://doi.acm.org/10.1145/2457317.2457374 (cit. on p. 32).

Gandrud, Christopher (2013). *Reproducible Research with R and R Studio*. CRC Press (cit. on p. 159).

Garijo, D. et al. (2014). "Workflow Reuse in Practice: A Study of Neuroimaging Pipeline Users". In: *e-Science (e-Science), 2014 IEEE 10th International Conference on*. Vol. 1, pp. 239–246. DOI: 10.1109/eScience.2014.33 (cit. on p. 52).

Gerlach, Wolfgang et al. (2014). "Skyport: Container-based Execution Environment Management for Multi-cloud Scientific Workflows". In: *Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds*. DataCloud '14. New Orleans, Louisiana: IEEE Press, pp. 25–32. URL: http://dx.doi.org/10.1109/DataCloud.2014.6 (cit. on p. 37).

Glatard, Tristan et al. (2015). "Reproducibility of neuroimaging analyses across operating systems". In: *Frontiers in Neuroinformatics* 9, p. 12 (cit. on pp. 140, 159).

Glavic, Boris and Klaus R Dittrich (2007). "Data Provenance: A Categorization of Existing Approaches." In: *BTW*. Vol. 7. 12. Citeseer, pp. 227–241 (cit. on pp. 23, 25).

Goldberg, Robert P. (1974). "Survey of Virtual Machine Research". In: *Computer* 7.9 (Sept. 1974), pp. 34–45. URL: http://dx.doi.org/10.1109/MC.1974.6323581 (cit. on p. 20).

Gómez-Pérez, José Manuel et al. (2013). "How Reliable is Your Workflow: Monitoring Decay in Scholarly Publications." In: Citeseer (cit. on p. 48).

Griphyn (2014). http://www.phys.utb.edu/griphyn/. Online; Last Visited on 12-Sept-2015 (cit. on p. 24).

Gronenschild, Ed H B M et al. (2012). "The effects of FreeSurfer version, workstation type, and Macintosh operating system version on anatomical volume and cortical thickness measurements." eng. In: *PLoS One* 7.6, e38234. DOI: 10.1371/journal.pone.0038234 (cit. on pp. 52, 106, 140, 158, 159).

Groth, P. et al. (2005). "Recording and Using Provenance in a Protein Compressibility Experiment". In: *Proceedings of the High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium*. HPDC '05. Washington, DC, USA: IEEE Computer Society, pp. 201–208. URL: http://dx.doi.org/10.1109/HPDC.2005.1520960 (cit. on p. 24).

Groth, Paul et al. (2009). "Pipeline-centric Provenance Model". In: *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*. WORKS '09. Portland, Oregon: ACM, 4:1–4:8. URL: http://doi.acm.org/10.1145/1645164.1645168 (cit. on pp. 31, 34, 105, 152).

Guo, Philip J. and Dawson Engler (2011). "CDE: Using System Call Interposition to Automatically Create Portable Software Packages". In: *Proceedings of the 2011 USENIX Annual Technical Conference*. USENIX'11. Portland, OR: USENIX Association. URL: http://dl.acm.org/citation.cfm?id=2002181.2002202 (cit. on p. 37).

Hasham, Khawar et al. (2011). "CMS workflow execution using intelligent job scheduling and data access strategies". In: *Nuclear Science, IEEE Transactions on* 58.3, pp. 1221–1232 (cit. on p. 106).

Hoffa, C. et al. (2008). "On the Use of Cloud Computing for Scientific Workflows". In: *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pp. 640–645. DOI: 10.1109/eScience.2008.167 (cit. on pp. 19, 105).

Howe, Bill (2012). "Virtual Appliances, Cloud Computing, and Reproducible Research". In: *Computing in Science Engineering* 14.4, pp. 36–41 (cit. on p. 48).

Imran, Muhammad and Helmut Hlavacs (2012). "Provenance in the cloud: Why and how?" In: *CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization*, pp. 106–112 (cit. on pp. 19, 22, 33, 34).

Imran, Muhammad and Helmut Hlavacs (2013). "Provenance Framework for the Cloud Infrastructure: Why and How?" In: *Advances in Intelligent Systems* 6 (1 2013) (cit. on p. 22).

Islam, Sidra (2010). "Provenance, Lineage, and Workflows". PhD thesis. Brown University, RI, USA (cit. on p. 28).

Janin, Yves, Cédric Vincent and Rémi Duraffort (2014). "CARE, the Comprehensive Archiver for Reproducible Execution". In: *Proceedings of the 1st ACM SIGPLAN Workshop on Reproducible Research Methodologies and New Publication Models in Computer Engineering*. TRUST '14. New York, NY, USA: ACM, 1:1–1:7. URL: http://doi.acm.org/10.1145/2618137.2618138 (cit. on pp. 36, 37, 45).

Juve, G. et al. (2009). "Scientific workflow applications on Amazon EC2". In: *E-Science Workshops, 2009 5th IEEE International Conference on*, pp. 59–66. DOI: 10.1109/ESCIW.2009.5408002 (cit. on pp. 10, 27, 57).

Juve, Gideon and Ewa Deelman (2010). "Scientific Workflows and Clouds". In: *Crossroads* 16.3 (Mar. 2010), pp. 14–18. URL: http://doi.acm.org/10.1145/1734160.1734166 (cit. on pp. 2, 5, 19, 62).

Juve, Gideon et al. (2013). "Characterizing and profiling scientific workflows". In: *Future Generation Computer Systems* 29.3, pp. 682–692 (cit. on pp. 104–107).

Kanwal, S. et al. (2015). "Challenges of Large-Scale Biomedical Workflows on the Cloud – A Case Study on the Need for Reproducibility of Results". In: *Computer-Based Medical Systems (CBMS), 2015 IEEE 28th International Symposium on*, pp. 220–225. DOI: 10.1109/CBMS.2015.28 (cit. on pp. 2, 44, 158).

Kim, Jihie et al. (2008). "Provenance Trails in the Wings-Pegasus System". In: *Concurr. Comput. : Pract. Exper.* 20.5 (Apr. 2008), pp. 587–597. URL: http://dx.doi.org/10.1002/cpe.v20:5 (cit. on p. 27).

Klinginsmith, J., M. Mahoui and Y.M. Wu (2011). "Towards Reproducible eScience in the Cloud". In: *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pp. 582–586. DOI: 10.1109/CloudCom.2011.89 (cit. on pp. 6, 39, 162).

Ko, Ryan KL, Peter Jagadpramana and Bu Sung Lee (2011). "Flogger: A file-centric logger for monitoring file access and transfers within cloud computing environments". In: *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*. IEEE, pp. 765–771 (cit. on p. 33).

KVM (2015). *Kernal Based Virtual Machine*. www.linux-kvm.org/page/MainPage. Online; Last Visited on 12-Sept-2015 (cit. on p. 18).

Lifschitz, Sergio, Luciana Gomes and Stevens K Rehen (2011). "Dealing with reusability and reproducibility for scientific workflows". In: *Bioinformatics and Biomedicine Workshops (BIBMW), 2011 IEEE International Conference on*. IEEE, pp. 625–632 (cit. on pp. 46, 49, 92).

Lim, Chunhyeok et al. (2010). "Prospective and retrospective provenance collection in scientific workflow environments". In: *Services Computing (SCC), 2010 IEEE International Conference on*. IEEE, pp. 449–456 (cit. on p. 17).

Ludäscher, Bertram et al. (2006). "Scientific Workflow Management and the Kepler System: Research Articles". In: *Concurr. Comput. : Pract. Exper.* 18.10 (Aug. 2006), pp. 1039–1065. URL: http://dx.doi.org/10.1002/cpe.v18:10 (cit. on p. 27).

Macko, Peter, Marc Chiarini and Margo Seltzer (2011). "Collecting Provenance via the Xen Hypervisor." In: USENIX (cit. on p. 33).

Maheshwari, Ketan et al. (2013). "Evaluating cloud computing techniques for smart power grid design using parallel scripting". In: *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, pp. 319–326 (cit. on p. 32).

Malawski, Maciej et al. (2015). "Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds". In: *Future Generation Computer Systems* 48.0. Special Section: Business and Industry Specific Cloud, pp. 1 –18. URL: http://www.sciencedirect.com/science/article/pii/S0167739X15000059 (cit. on p. 50).

McClatchey, Richard et al. (2013). "Providing traceability for neuroimaging analyses". In: *International Journal of Medical Informatics* 82.9, pp. 882 –894. DOI: http://dx.doi.org/10.1016/j.ijmedinf.2013.05.005 (cit. on p. 54).

McClatchey, Richard et al. (2014). "Provenance support for medical research". In: *5th International Provenance and Annotation Workshop (IPAW2014)* (cit. on p. 26).

McCusker, K and S Gunaydin (2015). "Research using qualitative, quantitative or mixed methods and choice based on the research". In: *Perfusion* 30.7, pp. 537–542. DOI: 10.1177/0267659114559116 (cit. on p. 8).

Mei, Lijun, W. K. Chan and T. H. Tse (2008). "A Tale of Clouds: Paradigm Comparisons and Some Thoughts on Research Issues". In: *Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference*. APSCC '08. Washington, DC, USA: IEEE Computer Society, pp. 464–469. DOI: 10.1109/APSCC.2008.168. URL: http://dx.doi.org/10.1109/APSCC.2008.168 (cit. on p. 1).

Meier, Wolfgang (2003). "eXist: An Open Source Native XML Database". In: *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*. London, UK, UK: Springer-Verlag, pp. 169–183. URL: http://dl.acm.org/citation.cfm?id=648032.744076 (cit. on pp. 26, 54).

Mell, Peter M. and Timothy Grance (2011). *SP 800-145. The NIST Definition of Cloud Computing*. Tech. rep. Gaithersburg, MD, United States (cit. on p. 1).

Missier, Paolo et al. (2012). "Golden trail: Retrieving the data history that matters from a comprehensive provenance repository". In: *International Journal of Digital Curation* 7.1, pp. 139–150 (cit. on p. 50).

Missier, Paolo et al. (2013). "Provenance and data differencing for workflow reproducibility analysis". In: *Concurrency and Computation: Practice and Experience*, n/a–n/a. URL: http://dx.doi.org/10.1002/cpe.3035 (cit. on pp. 35, 49, 82, 142, 157).

Moreau, Luc (2010). "The Foundations for Provenance on the Web". In: *Found. Trends Web Sci.* 2.2–3 (Feb. 2010), pp. 99–241. URL: http://dx.doi.org/10.1561/1800000010 (cit. on p. 2).

Munir, Kamran et al. (2013). "An Integrated e-Science Analysis Base for Computation Neuroscience Experiments and Analysis". In: *Procedia - Social and Behavioral Sciences* 73.0. Proceedings of the 2nd International Conference on Integrated Information (IC-ININFO 2012), Budapest, Hungary, August 30 – September 3, 2012, pp. 85 –92. DOI: http://dx.doi.org/10.1016/j.sbspro.2013.02.026. URL: http://www.sciencedirect.com/science/article/pii/S1877042813003194 (cit. on pp. 1, 52).

Munir, Kamran et al. (2014). "Provision of an integrated data analysis platform for computational neuroscience experiments". In: *Journal of Systems and Information Technology* 16.3, pp. 150–169 (cit. on p. 54).

Muniswamy-Reddy, Kiran-Kumar, Peter Macko and Margo Seltzer (2009). "Making a Cloud Provenance -aware". In: *First Workshop on on Theory and Practice of Provenance*. TAPP'09. San Francisco, CA: USENIX Association, 12:1–12:10. URL: http://dl.acm.org/citation.cfm?id=1525932.1525944 (cit. on pp. 30, 33).

Muniswamy-Reddy, Kiran-Kumar, Peter Macko and Margo Seltzer (2010). "Provenance for the cloud". In: *Proceedings of the 8th USENIX conference on File and storage technologies*. USENIX Association, pp. 15–14 (cit. on pp. 16, 19, 33).

Muniswamy-Reddy, Kiran-Kumar and Margo Seltzer (2010). "Provenance As First Class Cloud Data". In: *SIGOPS Oper. Syst. Rev.* 43.4 (Jan. 2010), pp. 11–16. URL: http://doi.acm.org/10.1145/1713254.1713258 (cit. on pp. 21, 30).

Myers, M.D. (1997). "Qualitative research in information systems". In: *MIS Quarterly* 21 (2 1997), pp. 241–242 (cit. on p. 8).

Ocana, K.A.C.S. et al. (2011). "Optimizing Phylogenetic Analysis Using SciHmm Cloud-based Scientific Workflow". In: *E-Science (e-Science), 2011 IEEE 7th International Conference on*, pp. 62–69. DOI: 10.1109/eScience.2011.17 (cit. on p. 120).

Oliveira, D. de et al. (2010). "SciCumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific Workflows". In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 378–385. DOI: 10.1109/CLOUD.2010.64 (cit. on p. 31).

Oxford (2015). - *define Provenance*. http://oxforddictionaries.com/definition/provenance. Online; Last Visited on 20-Sept-2015 (cit. on p. 2).

O'Brien, Rory (1998). *An Overview of the Methodological Approach of Action Research*. http://www.web.ca/ robrien/papers/arfinal.html. Online; Last Visited on 12-February-2016 (cit. on p. 8).

Pham, Quan, Tanu Malik and Ian Foster (2013). "Using Provenance for Repeatability". In: *Presented as part of the 5th USENIX Workshop on the Theory and Practice of Provenance*. Lombard, IL: USENIX. URL: https://www.usenix.org/conference/tapp13/using-provenance-repeatability (cit. on pp. 36, 45).

Pietri, Ilia et al. (2014). "A Performance Model to Estimate Execution Time of Scientific Workflows on the Cloud". In: *Proceedings of the 9th Workshop on Workflows in Support of Large-Scale Science*. WORKS '14. New Orleans, Louisiana: IEEE Press, pp. 11–19. URL: http://dx.doi.org/10.1109/WORKS.2014.12 (cit. on pp. 120, 158).

Prat, Nicolas and Stuart Madnick (2008). "Measuring data believability: A provenance approach". In: *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*. IEEE, pp. 393–393 (cit. on p. 6).

Raicu, I. et al. (2007). "Falkon: a Fast and Light-weight tasK executiON framework". In: *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*, pp. 1–12. DOI: 10.1145/1362622.1362680 (cit. on p. 16).

Ram, Sudha and Jun Liu (2007). "Understanding the Semantics of Data Provenance to Support Active Conceptual Modeling". English. In: *Active Conceptual Modeling of Learning*. Ed. by PeterP. Chen and LeahY. Wong. Vol. 4512. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 17–29. URL: http://dx.doi.org/10.1007/978-3-540-77503-4_3 (cit. on pp. 22, 55).

Ramakrishnan, Lavanya and Beth Plale (2010). "A Multi-dimensional Classification Model for Scientific Workflow Characteristics". In: *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science*. Wands '10. Indianapolis, Indiana: ACM, 4:1–4:12. URL: http://doi.acm.org/10.1145/1833398.1833402 (cit. on p. 107).

Redolfi, Alberto et al. (2009). "Grid infrastructures for computational neuroscience: the neuGRID example". In: *Future Neurology* 4.6, pp. 703–722 (cit. on p. 15).

Ricci, Robert et al. (2015). "Apt: A Platform for Repeatable Research in Computer Science". In: *SIGOPS Oper. Syst. Rev.* 49.1 (Jan. 2015), pp. 100–107. URL: http://doi.acm.org/10.1145/2723872.2723885 (cit. on pp. 38, 45).

Rimal, B.P., Eunmi Choi and I. Lumb (2009). "A Taxonomy and Survey of Cloud Computing Systems". In: *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, pp. 44–51. DOI: 10.1109/NCM.2009.218 (cit. on p. 18).

Roure, David De et al. (2011). "Towards the preservation of scientific workflows". In: *In Procs. of the 8th International Conference on Preservation of Digital Objects (iPRES 2011). ACM* (cit. on p. 48).

Rozsnyai, S., A. Slominski and Y. Doganata (2011). "Large-Scale Distributed Storage System for Business Provenance". In: *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pp. 516–524. DOI: 10.1109/CLOUD.2011.28 (cit. on p. 24).

Sakellariou, Rizos, Henan Zhao and Ewa Deelman (2010). "Mapping Workflows on Grid Resources: Experiments with the Montage Workflow". English. In: *Grids, P2P and Services Computing*. Ed. by Frédéric Desprez et al. Springer US, pp. 119–132. URL: http://dx.doi.org/10.1007/978-1-4419-6794-7_10 (cit. on pp. 104, 105).

Sandve, Geir Kjetil et al. (2013). "Ten Simple Rules for Reproducible Computational Research". In: *PLoS Comput Biol* 9.10 (Oct. 2013), e1003285. URL: http://dx.doi.org/10.1371%2Fjournal.pcbi.1003285 (cit. on pp. 45, 49).

Santana-Perez, Idafen and María S Pérez-Hernández (2015). "Towards reproducibility in scientific workflows: An infrastructure-based approach". In: *Scientific Programming* 2015 (cit. on p. 45).

Santana-Pérez, Idafen and MS Pérez-Hernández (2014). "Towards reproducibility in scientific workflows: An infrastructure-based approach". In: *IEEE Computing in Science & Engineering, p. submitted* (cit. on pp. 2, 45).

Santana-Perez, Idafen et al. (2014a). "A Semantic-Based Approach to Attain Reproducibility of Computational Environments in Scientific Workflows: A Case Study". English. In: *Euro-Par 2014: Parallel Processing Workshops*. Ed. by Luís Lopes et al. Vol. 8805. Lecture Notes in Computer Science. Springer International Publishing, pp. 452–463. URL: http://dx.doi.org/10.1007/978-3-319-14325-5_39 (cit. on pp. 38, 47).

Santana-Perez, Idafen et al. (2014b). "Leveraging Semantics to Improve Reproducibility in Scientific Workflows". In: *The reproducibility at XSEDE workshop* (cit. on pp. 38, 46, 47).

Scheidegger, Carlos et al. (2008). "Tackling the Provenance Challenge One Layer at a Time". In: *Concurr. Comput. : Pract. Exper.* 20.5 (Apr. 2008), pp. 473–483. URL: http://dx.doi.org/10.1002/cpe.v20:5 (cit. on p. 25).

Scheidegger, C.E. et al. (2007). "Querying and Creating Visualizations by Analogy". In: *Visualization and Computer Graphics, IEEE Transactions on* 13.6, pp. 1560–1567. DOI: 10.1109/TVCG.2007.70584 (cit. on p. 24).

SDSS (2014). http://www.sdss.org. Online; Last Visited on 12-Sept-2015 (cit. on p. 24).

Shamdasani, Jetendre, Andrew Branson and Richard McClatchey (2012). "Towards Semantic Provenance in CRISTAL". In: *Third International Workshop on the role of Semantic Web in Provenance Management (SWPM 2012)* (cit. on pp. 3, 19).

Simmhan, Yogesh and Beth Plale (2011). "Using Provenance for Personalized Quality Ranking of Scientific Datasets." In: *IJ Comput. Appl.* 18.3, pp. 180–195 (cit. on p. 50).

Simmhan, Yogesh L., Beth Plale and Dennis Gannon (2005). "A Survey of Data Provenance in e-Science". In: *SIGMOD Rec.* 34.3 (Sept. 2005), pp. 31–36. URL: http://doi.acm.org/10.1145/1084805.1084812 (cit. on pp. 2, 23).

SMS, Cruz et al. (2011). "Capturing Distributed Provenance Metadata from Cloud-Based Scientific Workflows". In: *Information and Data Management* 2, pp. 43–50 (cit. on p. 3).

Spulber, "Gabriela, Soheil Damangir and Lars-Olof Wahlund (2012). "D3.1: N4U Requirements for the Provision of Knowledge Management Services and Analysis Environment". In: (cit. on pp. 4, 53).

Spulber, Gabriela, Soheil Damangir and Lars-Olof Wahlund (2013). https://neugrid4you.eu/ -documents/10626/377198/D3.3.pdf, Online; Last Visited on 20-Sept-2015 (cit. on p. 4).

Stallings, William (2010). *Cryptography and Network Security: Principles and Practice*. 5th. Upper Saddle River, NJ, USA: Prentice Hall Press (cit. on p. 92).

Starlinger, Johannes et al. (2014a). "Layer Decomposition: An Effective Structure-based Approach for Scientific Workflow Similarity". In: *e-Science (e-Science), 2014 IEEE 10th International Conference on*. Vol. 1. IEEE, pp. 169–176 (cit. on p. 83).

Starlinger, Johannes et al. (2014b). "Similarity Search for Scientific Workflows". In: *Proceedings of the VLDB Endowment (PVLDB)*, p. 12 (cit. on p. 83).

Stevens, Robert D., Alan J. Robinson and Carole A. Goble (2003). "myGrid: personalised bioinformatics on the information grid". In: *Bioinformatics* 19 (suppl 1 2003), pp. i302–i304 (cit. on p. 26).

Stodden, Victoria (2011). https://web.stanford.edu/ vcs/talks/OAuofAOct252011-STODDEN.pdf. Online; Last Visited on 20-Sept-2015 (cit. on p. 35).
Strijkers, R. et al. (2010). "AMOS: Using the Cloud for On-Demand Execution of e-Science Applications". In: *e-Science (e-Science), 2010 IEEE Sixth International Conference on*, pp. 331–338. DOI: 10.1109/eScience.2010.15 (cit. on pp. 19, 20, 57).
Tan, Wang Chiew (2004). "Research Problems in Data Provenance." In: *IEEE Data Eng. Bull.* 27.4, pp. 45–52 (cit. on p. 23).
Tan, Yu Shyang et al. (2012). "Tracking of Data Leaving the Cloud". In: *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications* 0, pp. 137–144. DOI: http://doi.ieeecomputersociety.org/10.1109/TrustCom.2012.282 (cit. on pp. 33, 34).
Tannenbaum, Todd et al. (2002). "Beowulf Cluster Computing with Linux". In: Cambridge, MA, USA: MIT Press. Chap. Condor: A Distributed Job Scheduler, pp. 307–350. URL: http://dl.acm.org/citation.cfm?id=509876.509893 (cit. on pp. 20, 32, 57).
Vöckler, Jens-Sönke et al. (2011). "Experiences Using Cloud Computing for a Scientific Workflow Application". In: *Proceedings of the 2Nd International Workshop on Scientific Cloud Computing*. ScienceCloud '11. San Jose, California, USA: ACM, pp. 15–24. URL: http://doi.acm.org/10.1145/1996109.1996114 (cit. on pp. 5, 10, 16, 27, 47, 48, 56–58).
Vouk, M.A. (2008). "Cloud computing #x2014; Issues, research and implementations". In: *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*, pp. 31–40 (cit. on p. 3).
Wang, Jianwu and Ilkay Altintas (2012). "Early Cloud Experiences with the Kepler Scientific Workflow System". In: *Procedia Computer Science* 9.0. Proceedings of the International Conference on Computational Science, {ICCS} 2012, pp. 1630 –1634. DOI: http://dx.doi.org/10.1016/j.procs.2012.04.179 (cit. on p. 28).
Wilde, Michael et al. (2011). "Swift: A language for distributed parallel scripting". In: *Parallel Computing* 37.9, pp. 633–652 (cit. on pp. 31, 32, 152).
Williamson, Amelia (2014). *Feature - A side of cloud with your grid, ma'am?* http://www.isgtw.org/-?pid=1001800. Online; Last Visited on 12-Sept-2015 (cit. on pp. 19, 57).
Wolstencroft, Katherine et al. (2013). "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud". In: *Nucleic acids research*, gkt328 (cit. on p. 26).
Woodman, Simon et al. (2011). "Achieving Reproducibility by Combining Provenance with Service and Workflow Versioning". In: *Proceedings of the 6th Workshop on Workflows in Support of Large-scale Science*. WORKS '11. New York, NY, USA: ACM, pp. 127–136. URL: http://doi.acm.org/10.1145/2110497.2110512 (cit. on p. 49).
Workflow-definition (2015). *Workflow on the web*. http://www.globotron.com/htm/whitepapers/ -workweb.pdf. Online; Last Visited on 12-Sept-2015 (cit. on p. 15).
Xen (2015). www.xensource.com. Online; Last Visited on 12-Sept-2015 (cit. on p. 18).
Ylonen, Tatu (1996). "SSH–secure login connections over the Internet". In: (cit. on p. 32).
Zhang, Olive Qing et al. (2011). "How to track your data: The case for cloud computing provenance". In: *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE, pp. 446–453 (cit. on pp. 21, 33, 34).
Zhao, Jing, K. Gomadam and V. Prasanna (2011). "Predicting Missing Provenance Using Semantic Associations in Reservoir Engineering". In: *Semantic Computing (ICSC), 2011 Fifth IEEE International Conference on*, pp. 141–148. DOI: 10.1109/ICSC.2011.42 (cit. on p. 51).
Zhao, Jun et al. (2004). "Using Semantic Web Technologies for Representing E-science Provenance". English. In: *The Semantic Web – ISWC 2004*. Ed. by SheilaA. McIlraith, Dimitris Plexou-

sakis and Frank van Harmelen. Vol. 3298. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 92–106. URL: http://dx.doi.org/10.1007/978-3-540-30475-3_8 (cit. on p. 26).

Zhao, Xun et al. (2014a). "Liquid: A Scalable Deduplication File System for Virtual Machine Images". In: *Parallel and Distributed Systems, IEEE Transactions on* 25.5, pp. 1257–1266. DOI: 10.1109/TPDS.2013.173 (cit. on p. 48).

Zhao, Yong et al. (2011). "Opportunities and Challenges in Running Scientific Workflows on the Cloud". In: *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on*, pp. 455–462. DOI: 10.1109/CyberC.2011.80 (cit. on pp. 3, 17, 21, 30).

Zhao, Yong et al. (2014b). "Enabling scalable scientific workflow management in the Cloud". In: *Future Generation Computer Systems* 0, pp. –. URL: http://www.sciencedirect.com/science/article/pii/S0167739X14002179 (cit. on pp. 19, 22).

Appendix A

Workflow and Configuration files

A.1 Wrapper Service Configurations

```
UPLOAD_FOLDER='upload_wfs/'
SQLALCHEMY_DATABASE_URI='mysql://USER:XXXXXX@164.11.100.75/DATABASE'
#directory to store user files locally
WMS_DIR='peg_files'
#load worklfow management system plugin to process user request
WMS_PLUGIN=pegasus
```

Listing A.2: The configurations of WMS WrapperService

A.2 ReCAP Prototype Configurations

```
[cloud_settings]
swift_host=164.11.100.72
service_name=Compute Service
OS_USERNAME=mkhahmad
OS_PASSWORD=xxxxxxxx
OS_AUTH_URL=https://api.opensciencedatacloud.org:5000/sullivan/v2.0/tokens
OS_TENANT_NAME=mkhahmd
OS_REGION_NAME=RegionOne
#mapping types could be static,eager,lazy
MAPPING_TYPE=static
[storage_settings]
swift_host=164.11.100.72
```

```
OS_USERNAME=admin
OS_PASSWORD=xxxxxxxxx
OS_AUTH_URL=http://164.11.100.72:5000
OS_TENANT_NAME=admin
OS_REGION_NAME=UWE_Region
EC2_URL=http://cccs05.cccs.uwe.ac.uk:8773/services/Cloud
EC2_ACCESS_KEY=<EC2_ACCESS_KEY>
EC2_SECRET_KEY=<EC2_SECRET_KEY>
[wmsdb_settings]
user=pegUser
password=xxxxxxx
host=robb.cccs.uwe.ac.uk
port=3306
dburl=mysql+mysqlconnector://pegUser:xxxxxxx@164.11.100.75/pegasusdb
database=pegasusdb
[recapdb_settings]
user=CAPuser
password=xxxxxxxx
host=robb.cccs.uwe.ac.uk
port=3306
dburl=mysql+mysqlconnector://CAPuser:xxxxxxx@164.11.100.75/cloudprov
database=cloudprov
[WMS_settings]
wms_monitor=PegasusMonitor
wms_parser=PegasusParser
[WrapperService]
endpoint=http://robb.cccs.uwe.ac.uk:5000/service_wrapper/api/v1.0
service_user=khawar
service_password=XXXXX
[log_settings]
log_conf=/Users/khawar/Documents/CloudProv/MultiLayerProv/conf/logging.conf
```

Listing A.3: ReCAP prototype configurations

A.3 Sample Workflow used for Host availability Test

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated: 2014-11-31 21:35:57.407563 -->
<!-- generated by: khawar -->
```

```
<!-- generator: python -->
<adag xmlns="http://pegasus.isi.edu/schema/DAX"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation = "http://pegasus.isi.edu/schema/DAX
http://pegasus.isi.edu/schema/dax-3.4.xsd"
version="3.4"
name="DynamicReqWF">
<file name="wordlist">
  <pfn url="file:///opt/parallel_jobs/wordfile" site="local"/>
</file>
<executable name="dummyjob" namespace="dynamicreqwf" version="1.0"
arch="x86_64" os="linux" installed="true">
  <pfn url="file:///opt/parallel_jobs/peg_jobs/dynamic_req/dummyjob.py"
site="condorpool"/>
</executable>
<job id="ID0000001" namespace="dynamicreqwf"
name="dummyjob" version="1.0">
  <argument>2 -i <file name="wordlist"/> -o <file name="dummyout"/>
  </argument>
  <uses name="dummyout" link="output" register="true" transfer="true"/>
  <uses name="wordlist" link="input" register="true" transfer="true"/>
  <invoke when="start">
    /home/khawar/phdcode/dynamic_req_test/PegDynamicTest.py
  </invoke>
</job>
</adag>
```

Listing A.4: Single job sample workflow for the Host information availability time test

A.4 Example Flavours provided by Amazon EC2 and OSDC

Flavour	vCPU	RAM (GB)
t2.micro	1	1
t2.small	1	2
t2.medium	2	4
t2.large	2	8
m3.medium	1	3.75
m3.large	2	7.5

a: A subset of instance flavours provided by Amazon EC2

Flavour Name	VCPUs	RAM	Root Disk
m1.tiny	1	512MB	1GB
m1.small	1	2048MB	20GB
m1.medium	2	4096MB	40GB
m1.large	4	8192MB	80GB
m1.xlarge	8	16384MB	160GB

b: Instance flavours provided by OpenStack in Open Science Data Cloud (OSDC)

Figure A.1: Instance Flavour provided by Amazon EC2 and OSDC

Appendix B

Workflow Outputs

B.1 Collected Machine information XML for N4U

```
<?xml version="1.0" ?>
<MonitoringInfo>
<SysInfo>
<hostname>
srv16.fatebenefratelli.it
</hostname>
<hostip> 193.204.145.16 </hostip>
<machine> x86_64 </machine>
<version>
#1 SMP Wed Mar 12 14:06:15 CDT 2014
</version>
<platform>
Linux-2.6.18-371.6.1.el5xen-x86_64-with-redhat-5.10-Boron
</platform>
<system> Linux </system>
<architecture>('64bit', 'ELF')</architecture>
<uname>
('Linux', 'srv16.fatebenefratelli.it', '2.6.18-371.6.1.el5xen', '#1 SMP Wed Mar 12 14:06:15
    ↪ CDT 2014', 'x86_64', 'x86_64')
</uname>
</SysInfo>
<CPUInfo>
<model_name>
Intel(R) Xeon(R) CPU E5410  @ 2.33GHz
</model_name>
<cpu_cores>  1 </cpu_cores>
```

```
<cpu_speed>  2327.568 </cpu_speed>
</CPUInfo>
<Process>
<pID>3635</pID>
<pExe>sh</pExe>
<pCMDline>
sh -c ;/bin/bash neugridLauncher.sh
    ↪ /grid/vo.neugrid.eu/software/ExpressLane/Scripts/nG+FreeSurfer+5.3.0+ReconAll+v01.script
    ↪ 124 OTp-
    ↪ pZmU3aTkxOWk5ZTE4NWdpMWU9PDsxZjg8Ojg7PGhlOzRmPm9sZXtldjJlbHFlaERnaXZyMmd
    ↪ 16092014-153635
    ↪ lfn://grid/vo.neugrid.eu/home/DC=ch_DC=cern_OU=Organic_Units_OU=Users_CN=khawar_
    ↪ nG+FreeSurfer+5.3.0+ReconAll+v01 nG+FreeSurfer+5.3.0+ReconAll+v01
    ↪ nG+OASIS+OAS10002MR1+1T5+MPR+ORIG+VO1.nii.bz2 \$* >
    ↪ job.output.stdout 2>; job.output.stderr;
</pCMDline>
<pCreateTime>15:37</pCreateTime>
</Process>
</MonitoringInfo>
```

Listing B.5: Augmented provenance information for N4U using the ReCAP concept

B.2 ReconAll Output Snippet

Following are the two output snippets taken from ReconAll log files.

```
#######################################
program versions used
$Id: recon-all,v 1.379.2.73 2013/05/12 23:15:37 nicks Exp $
$Id: mri_motion_correct.fsl,v 1.14 2011/03/02 20:16:39 nicks Exp $
mri_convert -all-info
ProgramName: mri_convert  ProgramArguments: -all-info  ProgramVersion: $Name:
    ↪ stable5 $  TimeStamp: 2015/09/28-04:16:28-GMT  BuildTimeStamp: May 13 2013
    ↪ 16:24:28  CVS: $Id: mri_convert.c,v 1.179.2.7 2012/09/05 21:55:16 mreuter Exp $
    ↪ User: ubuntu  Machine: newfree.novalocal  Platform: Linux  PlatformVersion:
    ↪ 3.2.0-24-virtual  CompilerName: GCC  CompilerVersion: 40400
FLIRT version 5.5
$Id: talairach_avi,v 1.9 2011/03/02 18:38:06 nicks Exp $
```

Listing B.6: Showing timestamp in a log file (mri_convert) produced by ReconAll

```
#######################################
program versions used
$Id: recon-all,v 1.379.2.73 2013/05/12 23:15:37 nicks Exp $
$Id: mri_motion_correct.fsl,v 1.14 2011/03/02 20:16:39 nicks Exp $
mri_convert -all-info
ProgramName: mri_convert  ProgramArguments: -all-info  ProgramVersion: $Name:
    ↪ stable5 $  TimeStamp: 2015/09/29-05:11:24-GMT  BuildTimeStamp: May 13 2013
    ↪ 16:24:28  CVS: $Id: mri_convert.c,v 1.179.2.7 2012/09/05 21:55:16 mreuter Exp $
    ↪ User: ubuntu  Machine: newfree.novalocal  Platform: Linux  PlatformVersion:
    ↪ 3.2.0-24-virtual  CompilerName: GCC  CompilerVersion: 40400
FLIRT version 5.5
$Id: talairach_avi,v 1.9 2011/03/02 18:38:06 nicks Exp $
```

Listing B.7: Showing timestamp in a log file (mri_convert) produced by ReconAll

B.3 Montage Workflow Mosaic Output

This were the final output files produced by the montage workflows 533 (original execution) and 534 (reproduced execution).

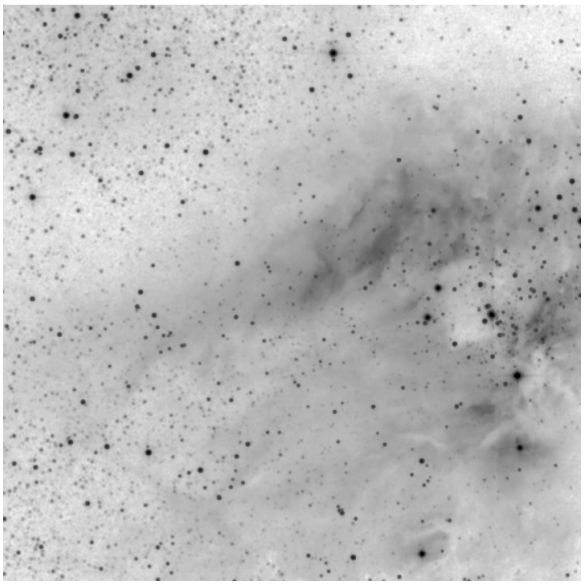


Figure B.1: The final Mosaic output (in JPEG) produced by workflow 533

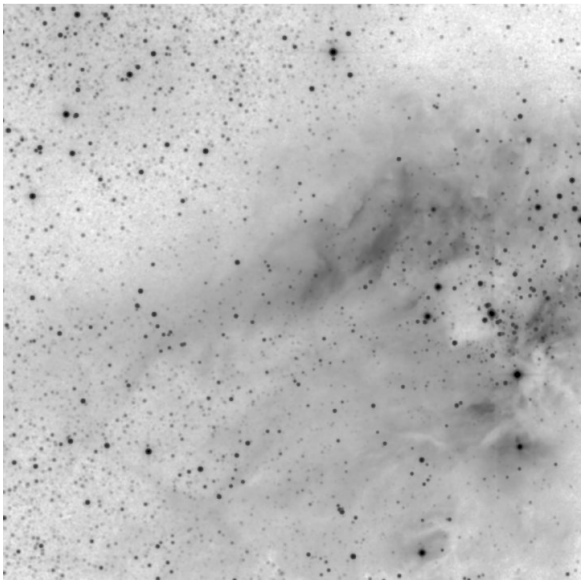


Figure B.2: The final Mosaic output (in JPEG) produced by workflow 534

B.4 ReconAll workflow outputs comparison

Table B.1: List of files related to area, thickness and volume of brain with same hash. Output Comparison of the ReconAll workflows 545 (original workflow) and 547 (reproduced workflow)

WfID 545		WfID 547	
MD5	**Filename**	**MD5**	**Filename**
002e45ac61cc2ef3d0b3378c01375ac2	rh.area.mid	002e45ac61cc2ef3d0b3378c01375ac2	rh.area.mid
0223ecd0515eb5d692d0175c5250cc6f	rh.smoothwm.BE.crv	0223ecd0515eb5d692d0175c5250cc6f	rh.smoothwm.BE.crv
0523150b4d43d3fd81127bf3a46bb01e	rh.smoothwm.S.crv	0523150b4d43d3fd81127bf3a46bb01e	rh.smoothwm.S.crv
080d7d3d2907c140332043a8b7865b6f	lh.area.pial	080d7d3d2907c140332043a8b7865b6f	lh.area.pial
0e5a110ae6227976a0769db0cc772f10	rh.smoothwm.K2.crv	0e5a110ae6227976a0769db0cc772f10	rh.smoothwm.K2.crv
11b18c14d61de4c3677c39e010bab788	rh.area.pial	11b18c14d61de4c3677c39e010bab788	rh.area.pial
11ed89688451463387120893ed3fa8b7	lh.smoothwm.C.crv	11ed89688451463387120893ed3fa8b7	lh.smoothwm.C.crv
19c7af71354ea6f2a0b4851b178c3229	rh.curv.pial	19c7af71354ea6f2a0b4851b178c3229	rh.curv.pial
1c7ffa13850857b55fec9d62484a70de	rh.defect_labels	1c7ffa13850857b55fec9d62484a70de	rh.defect_labels
21d84c199b3e353f292def20cffebb4d	lh.avg_curv	21d84c199b3e353f292def20cffebb4d	lh.avg_curv
2a85b0adef9a95f484f45032c12ee06c	rh.smoothwm.C.crv	2a85b0adef9a95f484f45032c12ee06c	rh.smoothwm.C.crv
2b25df937e11191accbf5bec25c95a65	lh.curv.pial	2b25df937e11191accbf5bec25c95a65	lh.curv.pial
3825532a13772e2d1cfb793b2f1b0db0	rh.avg_curv	3825532a13772e2d1cfb793b2f1b0db0	rh.avg_curv
39b52edefd42ac8f6e769ae77a282c67	lh.defect_chull	39b52edefd42ac8f6e769ae77a282c67	lh.defect_chull
41f34e5de25437316d4c15fc96153078	rh.jacobian_white	41f34e5de25437316d4c15fc96153078	rh.jacobian_white
428304b8af158d4b8db402d9536a752d	lh.smoothwm.K.crv	428304b8af158d4b8db402d9536a752d	lh.smoothwm.K.crv
447beae68b0b9ae414ef6fad0abbad77	lh.jacobian_white	447beae68b0b9ae414ef6fad0abbad77	lh.jacobian_white
4a06a07d636d7b57101d13c73bd4ce50	rh.thickness	4a06a07d636d7b57101d13c73bd4ce50	rh.thickness
4a9ffb4c636d33a582b3aab798d8e4dd	lh.thickness	4a9ffb4c636d33a582b3aab798d8e4dd	lh.thickness

4bf618cedcec481f5b61ef99c2415f0e	lh.curv	4bf618cedcec481f5b61ef99c2415f0e	lh.curv
57a86ac2bd7981f89da09175a0f0d7df	rh.sulc	57a86ac2bd7981f89da09175a0f0d7df	rh.sulc
5c26999a001391f28482c04345bd4ef4	lh.defect_labels	5c26999a001391f28482c04345bd4ef4	lh.defect_labels
6855d1055f2a61466b622fd80fece2fd	lh.smoothwm.K1.crv	6855d1055f2a61466b622fd80fece2fd	lh.smoothwm.K1.crv
707114ad091a50cf02a43b9d2bd46b8d	rh.defect_borders	707114ad091a50cf02a43b9d2bd46b8d	rh.defect_borders
75779bc92369df9c51a9a062b662f88e	lh.smoothwm.FI.crv	75779bc92369df9c51a9a062b662f88e	lh.smoothwm.FI.crv
7a968b0ce5e27b5855d5a1ed4f021fd0	lh.area.mid	7a968b0ce5e27b5855d5a1ed4f021fd0	lh.area.mid
802a03b653471c964610563d34655290	lh.volume	802a03b653471c964610563d34655290	lh.volume
82c3e417144161fde4c52cdf0435cc98	rh.smoothwm.H.crv	82c3e417144161fde4c52cdf0435cc98	rh.smoothwm.H.crv
8849f3c4677fdd6571d8b8de9bba2b80	rh.defect_chull	8849f3c4677fdd6571d8b8de9bba2b80	rh.defect_chull
88d2fdf775ee1e2d8246cfd8ecdf461d	rh.smoothwm.K.crv	88d2fdf775ee1e2d8246cfd8ecdf461d	rh.smoothwm.K.crv
8dbe3d2c53cd505f88eba4827537877b	lh.smoothwm.H.crv	8dbe3d2c53cd505f88eba4827537877b	lh.smoothwm.H.crv
aa625fdd9b4edc07d089126d687a477c	lh.area	aa625fdd9b4edc07d089126d687a477c	lh.area
aa6e7ab79c17838ea8a66d7201329187	lh.sulc	aa6e7ab79c17838ea8a66d7201329187	lh.sulc
b25ce33ea2a30bc57d167c83bdb72663	rh.volume	b25ce33ea2a30bc57d167c83bdb72663	rh.volume
b3c5b784af616729b2c3c3a4ccec1d0c	rh.smoothwm.K1.crv	b3c5b784af616729b2c3c3a4ccec1d0c	rh.smoothwm.K1.crv
d448c8b55632df7e03566f4e547ff93b	rh.smoothwm.FI.crv	d448c8b55632df7e03566f4e547ff93b	rh.smoothwm.FI.crv
f342e67f5b44cd8a74d9926368511f2e	rh.area	f342e67f5b44cd8a74d9926368511f2e	rh.area
f7047ed58ddc339e0feca54b1b2e133e	rh.curv	f7047ed58ddc339e0feca54b1b2e133e	rh.curv
f880d8c626ee1da637d63958d489525d	lh.defect_borders	f880d8c626ee1da637d63958d489525d	lh.defect_borders
fc9b9af314fff41d121dc2b604456d72	lh.smoothwm.BE.crv	fc9b9af314fff41d121dc2b604456d72	lh.smoothwm.BE.crv
7ea45ad8602d746844fd7bb3821654c7	rh.curv.stats	7ea45ad8602d746844fd7bb3821654c7	rh.curv.stats
a78d4ff457bba34aeb5d68fe06919071	lh.curv.stats	a78d4ff457bba34aeb5d68fe06919071	lh.curv.stats

B.5 Provenance graphs of workflows 132 and 134

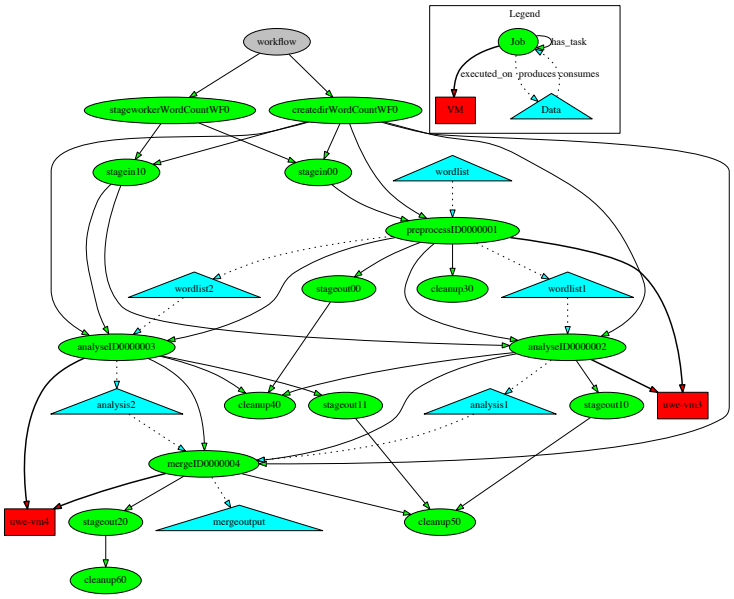


Figure B.3: The provenance graph of workflow 132

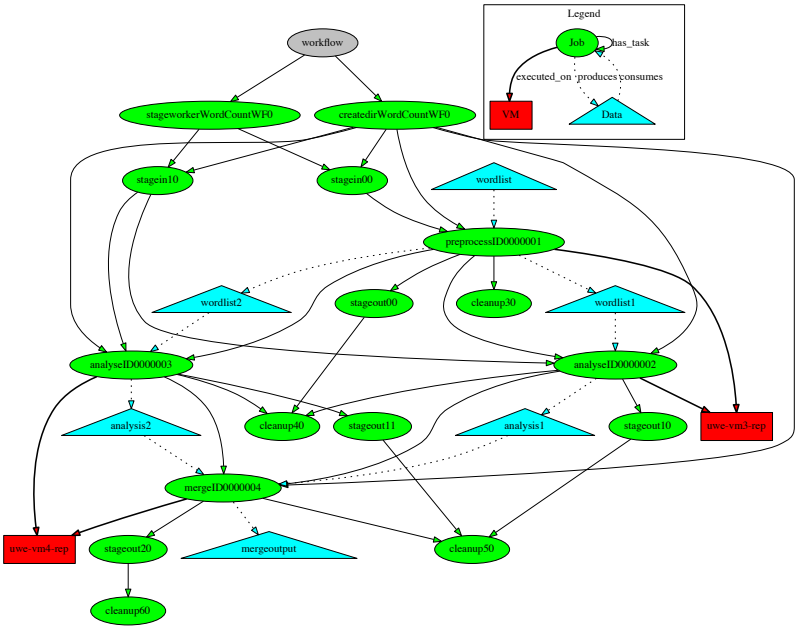


Figure B.4: The provenance graph of workflow 134

[File Creation Timestamp, phd_thesis.tex, 04/05/2016, 16:52]