Provenance-Driven Diagnostic Framework for Task Evictions Mitigating Strategy in Cloud Computing

Abdulaziz Mohammed N. Albatli

Submitted in accordance with the requirements for the degree of Doctor of Philosophy

UNIVERSITY OF LEEDS

The University of Leeds
School of Computing

January 2017

Intellectual Property and Publication Statements

The candidate confirms that the work submitted is his/her own, except where work which has formed part of jointly-authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

Albatli, A., McKee, D., Townend, P., Lau, L., Xu, J. (2017) **PROV-TE: A Provenance-Driven Diagnostic Framework for Task Eviction in Data Centers**. Proceedings of the 3rd IEEE International Conference on Big Data Computing Service and Applications (IEEE BigDataService 2017). (Accepted)

My contribution in this jointly authored publication was developing the diagnostic algorithms, designing the simulation setup, preparing the environment for evaluation, developing three models for task eviction behaviours, and conducting the analysis. David McKee contributed by providing the simulation tool and development guidance. The paper was reviewed editorially by David McKee, Paul Townend, Lydia Lau, and Jie Xu. The content of this paper is related to Chapters 2, 3, 4 and 6.

Albatli, A., Lau, L., Xu, J. (2014) **Application of PROV Model for Modeling a VM Overload Mitigating Strategy: Task Eviction**. Provenance Analytics Workshop. Provenance Week 2014.

My contribution in this jointly authored publication was conducting the analysis on Google 29-day dataset and proposing the early version of the novel provenance-driven diagnostic framework and discussing its potential contribution to the implementation of task eviction strategy for machine overload mitigation. Lydia Lau and Jie Xu provided structural feedback and helped editorially. The work relates to Chapters 2, 3, 4 and 5.

AlJahdali, H., Albatli, A., Garraghan, P., Townend, P., Lau, L., Xu, J. (2014) **Multi-Tenancy in Cloud Computing**. Proceedings of the 8th IEEE International Symposium on Service-Oriented System Engineering (SOSE 2014).

My contribution in this jointly authored publication was to understand and analyse Google 29-day dataset which was reflected in the introductory section of the paper. This was a preparatory step that helped conducting this research. Hassain AlJahdali did and wrote most of the publication, Peter Garraghan provided the selected log file of the dataset which I have analysed. Lydia Lau, Paul Townend, and Jie Xu provided structural feedback and helped editorially. The work relates to Chapters 4 and 5.

Acknowledgments

Without a doubt, first and foremost, my deepest appreciation and gratitude goes to my supervisors Dr. Lydia Lau and Prof. Jie Xu for their unconditional support and guidance, continuous and constructive feedback, and for shining the light on the path of my PhD journey. Thank you and God bless.

I thank my thesis examiners Dr. Ligang He and Dr. Karim Djemame for their excellent feedback and comments given in the examination.

I would like to thank my nephews, Ramzi and Faris, for their advice and initiative that honestly have let me start my unplanned, adventures, and amazing 12-year journey. You two have planted the seed that opened the doors to a better future. I will always be indebted to you both and forever be thankful.

I thank my friends and colleagues for their help in this PhD journey and for being there. Dave, David, Dimoklis, Django, Hussain, Ibrahim, Ismael, Judi, Khalid, Marwan, Paul, Peter, Richard, Silvana, and Vania. I cannot thank you enough.

Last but not least, I thank the Ministry of Education of the Saudi government for providing me a generous scholarship that gave me the opportunity to achieve my degrees, BSc, MSc and starting this PhD, abroad. Also, I thank Shaqra University for extending this opportunity to achieve my PhD degree.

Dedication

My sincere appreciations and heartfelt thank you goes to my parents, Munirah and Mohammed, and my siblings, Tarfah and Abdulmajeed, for their unconditional love and support not only during the years of my PhD but also during my expat life for the past 12 years. Without you, I would not have reached and accomplished this milestone. This work is dedicated to you.

I also thank my beloved wife, Daliyah Abuabah, whose divine love and encouragement can’t be overlooked. Thank you for being by my side. You are a true gift to my life. This work is dedicated to you too.

Abstract

Cloud computing is an evolving paradigm. It delivers virtualized, scalable and elastic resources (e.g. CPU, memory) over a network (e.g. Internet) from data centres to users (e.g. individuals, enterprises, governments). Applications, platforms, and infrastructures are Cloud services that users can access. Clouds enable users to run highly complex operations to satisfy computation needs through resource virtualization. Virtualization is a method to run a number of virtual machines (VM) on a single physical server. However, VMs are not a necessity in the Clouds. Cloud providers tend to overcommit resources, aiming to leverage unused capacity and maximize profits. This over-commitment of resources can lead to an overload of the actual physical machine, which lowers the performance or lead to the failure of tasks due to lack of resources, i.e. CPU or RAM, and consequently lead to SLA violations. There are a number of different strategies to mitigate the overload, one of which is VM task eviction.

The ambition of this research is to adapt a provenance model, PROV, to help understand the historical usage of a Cloud system and the components contributed to the overload, so that the causes for task eviction can be identified for future prevention. A novel provenance-driven diagnostic framework is proposed. By studying Google's 29-day Cloud dataset, the PROV model was extended to PROV-TE that underpinned a number of diagnostic algorithms for identifying evicted tasks due to specific causes.

The framework was implemented and tested against the Google dataset. To further evaluate the framework, a simulation tool, SEED, was used to replicate task eviction behaviour with the specifications of Google Cloud and Amazon EC2. The framework, specifically the diagnostic algorithms, was then applied to audit the causes and to identify the relevant evicted tasks. The results were then

analysed using precision and recall measures. The average precision and recall of the diagnostic algorithms are 83% and 90%, respectively.

Table of Contents

List of Acronyms

Ag	Agent
CbSP	Cloud-based Service Providers
CPN	Coloured Petri Net
CPU	Central Processing Unit
CSV	Comma-Separated Values
DCIM	Data Centre Infrastructure Management
DCSim	Data Centre Simulation
DTaP	Distributed Time-aware Provenance
EC2	Elastic Compute Cloud
EDT	Eastern Daylight Time
ER	Entity Relationship
FN	False Negative
FP	False Positive
IaaS	Infrastructure-as-a-Service
ID	Identification
IT	Information Technology
JE	Job Events
LIFO	Last-In-First-Out
MA	Machine Attributes
MDC	Machines Decreased Capacity table
MDCSim	Multi-tier Data Centre Simulation
ME	Machine Events
MEM	Memory
NGO	Non-Government Organization
NIST	National Institute of Standards and Technology
OCR	Over-Commitment Ratio
OPM	Open Provenance Model
OS	Operation System
OT	Overload Table
PaaS	Platform-as-a-Service
PCPU	Physical Central Processing Unit
PET	Priority of Evicted Tasks table
PM	Physical Machine
PROV-DM	The PROV Data Model
PROV-N	The Provenance Notation

PROV-O	The PROV Ontology
PROV-TE	PROV Task Eviction
QoS	Quality of Service
RAM	Random-Access Memory
RDF	The Resource Description Framework
RMI	Removed Machine IDs table
SaaS	Application-as-a-Service
SEED	Simulation EnvironmEnt Distributor
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SPADE	Support for Provenance Auditing in Distributed Environments
SQL	Structured Query Language
Std Dev	Standard Deviation
STRAPP	Trusted Digital Spaces through Timely Reliable and Personalised Provenance
TC	Task Constraint
TE	Task Eviction
TEv	Task Events
TN	True Negatives
TP	True Positives
TSR	Task Submission Rate
TU	Task Usage
UML	Unified Modelling Language
UTC	Coordinated Universal Time
VCPU	Virtual Central Processing Unit
VMemory	Virtual Memory
VM	Virtual Machine
VMM	Virtual Machine Manager
W3C	World Wide Web Consortium
WS-S	Web Service Security
XML	Extensible Markup Language
YANS	Yet Another Network Simulator

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Motivation

In Cloud computing, virtual machine over-commitment is widely implemented among Cloud providers in order to maximize profits and to utilize resources [1]. Some researchers use other names for this mechanism, such as overbooking, oversubscription and over-allocation [1]–[3]. Over-commitment means that for any given physical machine, Cloud providers allocate more virtual capacity than the actual capacity on the physical machine. Over-commitment can make better use of resources, namely CPU, memory, storage and network in the Cloud [4]. These resources have been found to be under-utilized because users tend to over-estimate their needs so they request more resources [5].

Every Cloud provider has specific policies to determine the amount of over-commitment ratio [1]. For example, if the ratio is 2 then a server with a capacity of 50 units can accept 100 units. In reality, requirements such as QoS (Quality of Service) are taken into consideration by Cloud providers. For any application, the QoS is defined on a per-VM basis. It is an assurance that the required resources and parameters would be fully supported, such as levels of performance and availability. In addition, the SLA (Service Level Agreement) is a service contract between a Cloud provider and a Cloud user which includes the QoS parameters that guarantees the quality of service required.

Over-commitment runs the risk of not meeting SLAs [6] by breaching key and agreed performance metrics, such as response time, execution time, and latency. The ability to guarantee the continuous availability of the agreed levels

of resources, e.g. memory or CPU, could also be affected as a consequence of lack of over-commitment administration in a virtualized environment.

In addition, over-commitment of resources, if not managed carefully, can cause overload on the physical machines, which has a deteriorating effect on the performance and availability of the Cloud service. For example, even though that 88% of memory overloads are transient and lasts for less than 2 minutes [2], it is considered a massive drawback and can still violate the SLAs and QoS agreements. The sensitivity of memory overload is high because it can lead to halts in the system or service. The provider in this case may have to compensate the client.

In order to summarize over-commitment and overload, consider *R* as a resource such as CPU, *C(R)* as its physical capacity such as 10 units, *Rreq* as the total approved requests to *R*, and *Ract* as the total consumption actually needed to *R.* Over-commitment occurs when the total approved requests to R, such as 15 units exceeds the physical capacity of R, 10 units, (*Rreq* > *C(R)*). Overload occurs when the total consumption of R, such as 11 units, exceeds the physical capacity of R, 10 units, (*Ract* > *C(R)*). Ideally, it is expected that the actual consumption of a resource does not exceed its actual physical capacity which at the same time is less than the total approved requests (*Ract* <= *C(R)* < *Rreq*).

To help providers not to violate the SLAs in terms of, for example, performance and uptime, there are a number of different strategies to mitigate the overload which aim to provide a continuous space and computing power for existing tasks to perform, namely; Resource Stealing, Quiescing, Live Migration, Streaming Disks, Network Memory, and Task Eviction [2], [4], [7], [8]. These existing strategies for mitigating overloads are largely reactive, i.e. after an overload has

taken place, the system will act upon it. The proposed framework in this research is a step towards a proactive preventative system.

Google, a Cloud service provider, implemented the mitigating strategy Task Eviction in its data centres [7]. Google stated a variety of causes for any overload scenario to occur [7]. This research is motivated to find a method to systematically identify the causes of an overload and their extent of impact and explain how is different from the current diagnostics methods, such as fault tolerance techniques. Instead of only mitigating the current overload, understanding the causal relationships between cause and effect may help identify preventative actions and reduce future occurrences of overload.

1.2 Research Focus

Over-commitment is facilitated by the use of virtualization technologies. Thus, raw data from the virtualization layer, such as physical machines' resources uptime, is important for auditing. For example, when one physical unit of CPU is virtualized to 10 units, the failure of the physical unit means all virtualized units will fail as a result which then leads to an overload on the servers. This research focuses on: the behaviour of the physical machines resources uptime (CPU, Memory, and disk), the behaviour of users in terms of their requests of resources, and the usage of the allocated resources. The use of PROV model [9], one of provenance models explained in section 2.3.2, to add reasoning power and meaning to logged usage data might lead to the exploration of reasons and causes of overload. Analysis of provenance information of a given task would pave the way to extract knowledge from usage data that was not identified using the standard logging system. For example, STRAPP project [10], explained in section 3.3.2, has sparked the motivation of using provenance data.

The proposed provenance-driven diagnostic framework includes three parts. The first part is the PROV-TE model, which is an application of the provenance model PROV over a Cloud infrastructure that mitigates overloads by applying Task Eviction strategy. As presented in Chapter 4, PROV-TE has been built after examining and filtering the raw data of a datacentre. The second part prepares the platform for queries and the third part develops the algorithms and the diagnostic analysis. Each algorithm identifies the link between a specific cause and the evicted task(s). Alternative provenance and non-provenance based methods are presented in sections 3.3.3 and 3.2.5.

Calculating the extent of every cause can determine the most dominant cause of an overload in a datacentre. The general assumption of this framework is that overload could be caused by a variety of reasons which include, but not excluding, both the user and the provider. The framework is a structure that serves the diagnostic investigations on Task Evictions in Cloud data centres.

1.3 Research Aim and Questions

The aim of this research is to produce a provenance-driven diagnostic framework that examines the historical data and finds the causes of an overload. This framework will utilize and extend a provenance model: PROV, which is a W3C family of documents that define an abstract standardized model, corresponding serializations and other supporting definitions [11]. The framework uses light weight semantics. The challenge that faces this research is to understand the meaning and relationship captured in the raw data that have not been known previously. The purpose of making use of light weight semantics following PROV is to add meaning and reasoning power in terms of connecting the collected raw data as nodes and edges. Thus, a diagnostic framework to provide the reasons

and causes of an overload can be developed. Although there are strategies to mitigate the overload, there is no approach in the literature to audit the causes by considering the provenance of the dataset that captures the behaviour of the systems and the users' usage.

In light of this, the research questions and the objectives for each question are:

1. How to formulate a suitable diagnostic provenance model that will help check the causes of overload in a Cloud platform?

    a. To define the purpose of this underlying provenance model.
    b. To find how the proposed model would add reasoning power to the raw data in Cloud environments.
    c. To compare the proposed model different with other techniques, assess the added-value, and illustrate how it is different from a simple ER model.

2. How to operationalize the model?

    a. To develop queries that could provide diagnosis.
    b. To reflect on the lessons learned from testing on Google dataset.

3. How to validate the model?

    a. To test the accuracy of PROV-TE.
    b. To assess the reliability of the model and the diagnostic algorithms.

1.4 Research Methodology

In general, there are two research approaches to data collection and analysis, quantitative and qualitative [12]. According to the literature, there are three types of methodologies applied in distributed systems, namely Prototyping, Simulation and Mathematical Modelling [13]–[15].

The research approach followed in this thesis is based on quantitative analysis and includes the use of simulation, prototyping and repeatable empirical experiments as follows:

- *Identification* of the mitigating strategies of an overload and provenance models used in a datacentre through an extensive literature review.
- *Analysis and characterization of the overload problems and task eviction causes in Google's datacentre using a 29-day Cloud dataset*.
- *Formulation of a provenance model to audit the impact of the causes of overload.* This is the first case study using Google Cloud dataset for learning. After examining the dataset, the abstract model of PROV was applied which led to development of PROV-TE model. Then, 10 different diagnostic algorithms were developed, each look at the data from a different perspective with the aim of identifying and pinpointing the evicted tasks. Both the model and the algorithms have gone through two iterations of development.
- *Validation of the developed provenance-driven framework on a simulated dataset for evaluation.* This is the second case study using a simulation tool, SEED, to simulate the overload behaviour in a datacentre. The generated datasets were then passed through the diagnostic framework. This application is an assessment of the reliability and the transferability of the framework.

1.5 Potential Areas for Contribution

The major contributions of this thesis is summarized below. An in depth discussion will be presented in Chapter 6.

- *A provenance framework that acts as a diagnostic tool to find the causes of an overload in the Clouds by two steps. First, extending the PROV model to represent a task eviction mitigating strategy. Second, identifying attributes relevant to the strategy,* related to research question number 1.
- *A computational version of the model for reasoning. Developing algorithms to find the cause-effect relationship between causes and tasks (identifying the evicted tasks because of each cause),* related to research question number 2.
- *The modelling of Task Eviction behaviors in a Cloud datacentre with provenance data using a simulation tool*. Demonstrating methods of simulating task evictions, related to research question number 3.

1.6 Thesis Outline

A summary of the remaining chapters is as follows:

- **Chapter 2** discusses the state of the art with respect to provenance in Cloud computing. It presents the concept of the Cloud in computing and the concept of provenance and related background. In addition, it highlights the benefits of provenance and provenance models in the Clouds.
- **Chapter 3** presents related work in terms of Cloud Computing and Provenance. The mechanism of over-commitment of resources leveraging virtualization. In addition, it highlights the current use of provenance and provenance models in the Clouds.
- **Chapter 4** presents the analysis and characterization of Google's 29-day Cloud dataset. Also, it demonstrates the extension of PROV model

and the diagnostic algorithms to fit a Cloud datacentre using Google dataset for learning.

- **Chapter 5** illustrates the application of PROV-TE model over Google dataset and shows the computational version of the model and its application.
- **Chapter 6** shows the framework's evaluation using a simulation tool, SEED. 15 simulated datasets have been generated to test the precision and recall of the framework.
- **Chapter 7** presents the conclusion of this thesis and provides the future work related to this area of study.

Chapter 2

General Background in Cloud Computing and Provenance

2.1 Introduction

This chapter provides the broad context of this research. To better understand the scope of this research, the background of Cloud computing is presented and discussed. The concept of resources virtualization is then described. The differences between Cloud and traditional data centres are drawn. Then, the concept of data provenance as well as its uses and benefits are presented.

2.2 Cloud Computing

Cloud computing is a rapid evolving paradigm [16]. It delivers virtualized, scalable, dynamic, pooled and elastic resources (i.e. CPU, memory) over a network (i.e. Internet) from off-site data centres to the users (i.e. individuals, enterprises, governments) [17]. Besides, it has been stated that Cloud computing facilitates the transformation of the way in which the IT utility is delivered in a wide variety of organizations [18]. Further, there is not yet an agreed definition of Cloud computing in the literature. There are more than 20 definitions covering various aspects of Cloud computing [19]. The most used definition in the literature is issued by the National Institute of Standards and Technology (NIST) [20] stating that

> "*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*.",

which also is going to be used along this research. In addition, Cloud computing encompasses the applications that are delivered as services and also the hardware and software that support the delivery of these services [21], [22]. In order to guarantee consumers' satisfaction, reasonable levels of Quality of Service (QoS) must be established and maintained which is a pivotal feature when providing such dynamic services. Furthermore, the technology of Cloud computing has unique characteristics comes in a number of types (delivery models) and deployment modes. All would be discussed in the next sub-sections, respectively.

2.2.1 Cloud Computing Characteristics

NIST [20] defines five unique characteristics that distinguish Cloud computing from other distributed system technologies which are described as follow:

- *On-demand self-service*: Cloud self-service interfaces provide mechanisms for the management of the entire service delivery lifecycle. Consumers can request, utilize and manage the computing resources such as network bandwidth and CPU power on-demand and without the need to interact with service administrators or providers.
- *Broad network access:* Cloud computing services are delivered over standard network protocols and accessed through heterogeneous platforms and devices used by consumers such as mobiles, laptops, tablets and workstations. This allows Cloud providers the capability and capacity to deliver wide range of services of different kinds to their consumers whose devices are connected through networks.
- *Resource pooling:* Cloud providers can pool their computing resources to multiple users (consumers) utilizing a multi-tenant model. These

resources are dynamically assigned and reassigned depending on the consumers' demands. Cloud services are provided with a sense of location independence where consumers generally have no control or knowledge of the exact location of the provided resources. However, depending on the provider's policies, the consumer can specify the location at a higher level of abstraction such as datacentre, city, or country.

- *Rapid elasticity:* Elasticity is the system's ability to add and remove computing capacity from a computing environment. Cloud providers can scale their resources in and out based on consumers' demands which saves costs. Elasticity of resources can be achieved in two ways: vertically by increasing or decreasing Virtual Machine's (VM) resources (scale-up) and horizontally by increasing or decreasing the number of VMs (scale-out). This makes the resources appear to be unlimited to the consumers.
- *Measured service:* Cloud resources usage can be monitored, reported and managed on a transparent manner for both the provider and the consumer. Consumers are billed on a pay-per-use basis which means that consumer are charged according to their actual resources usage. This allows the providers to monitor the usage patterns which can lead to improvement to the Cloud environment productivity and resources provision enhancement.

2.2.2 Cloud Actors

There are five Cloud actors (stakeholders) according the NIST Cloud computing Reference Architecture [23], as shown in figure 2.1:

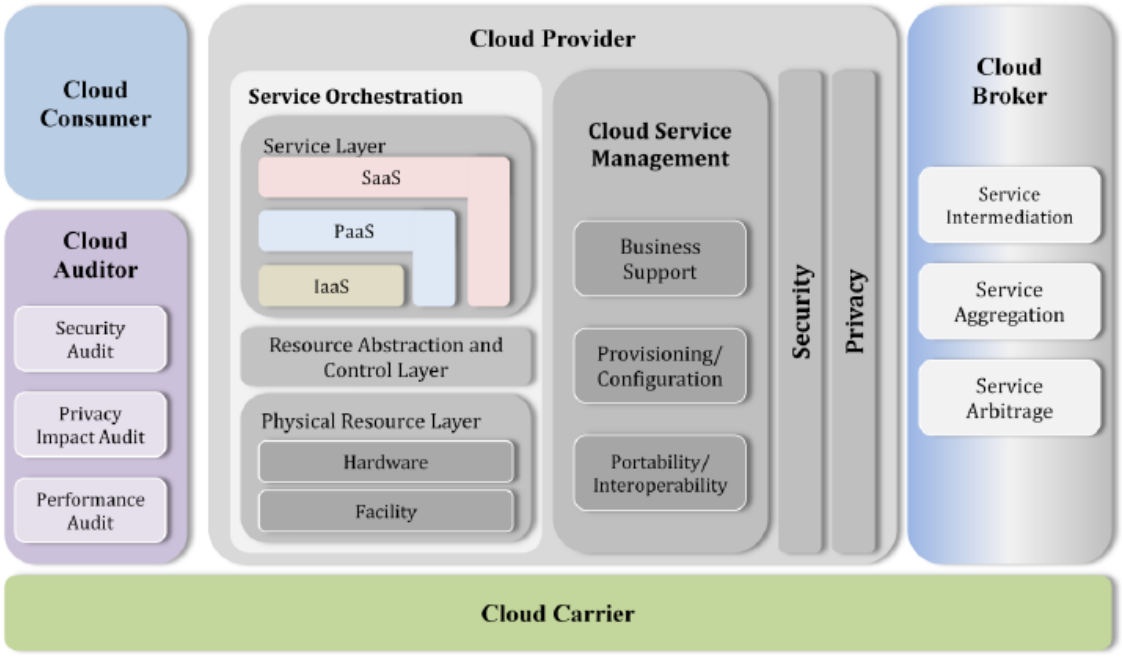

Figure 2.1 NIST Cloud Conceptual Reference Model [23]

- *Clouds Carriers* are the intermediate entities that guarantee faultless service distribution by providing transport and connectivity amongst other Cloud actors. They provide access to the their services through telecommunication, network and other devices which the other Cloud actors can use [17]. Besides, ensuring the distribution of the services is undertaken by telecommunication and network providers or transport agents. Those agents are defined as business organizations that provide the underlying physicality of the network. In addition, the Service Level Agreements (SLAs) are usually set up by the Cloud Provider with the Cloud Carrier to guarantee a consistent and dedicated delivery of Cloud services to fulfil the level of SLAs offered to consumers.
- *Cloud Brokers* provide negotiations between Cloud Providers and Cloud Consumers. The brokers' services can be categorized into two distinctive categories [17]. In the first category, the brokers deal with the relationships between Cloud providers and Cloud consumer. In addition,

the ownership and maintenance of the Cloud is the providers' responsibility not the broker. For instance, a consumer may reach a broker for a consultancy on the most suitable Cloud provider. In the second category, the broker can add additional services to the Cloud providers' application, platform, or infrastructure. Their aim is usually enhancing and adding more security components to the actual services which the providers are lacking.

- *Cloud Auditors* undertake an independent assessment of other Cloud actors and services. The aim of the audit is to confirm the expected/agreed level of standards in a number of dimensions, such as security, performance. The result of the audit is a certificate that has an influence of the consumers' choice of Cloud providers [17].
- *Cloud Providers* are responsible for the hosting and maintenance of the Cloud's infrastructure along with the provided services to consumers and brokers. The Cloud Provider is usually an individual or an organization. Cloud providers can undertake their activities in the following areas; privacy and security, Cloud service management, service orchestration, and service deployment. Usually, the providers' consumers (individuals or brokers) can become providers in specific cases [17].
- *Cloud Consumers* are the prime actor of Cloud computing. Habib in [17] identified two categories of Cloud consumers; end users, and Cloud-based service providers (CbSP). End users use the provided service to satisfy their goals (i.e. business targets). However, CbSPs offer (resell) the services they acquired, that are completely hosted in the Cloud, to their consumers. To distinguish between CbSPs and Cloud Brokers,

CbSP rely on the services they offer to build their own business model, while Cloud Brokers only offer extra add-on to the actual service.

2.2.3 Cloud Delivery Models

Implementations of Cloud computing can be categorized according to the service delivery model (Software, Platform, and Infrastructure) and according to the deployment mode (Public, Private, Community and Hybrid/Federated) [20], [24]–[27]. Each delivery model, shown in figure 2.2, has a different set of responsibilities and functionalities for both the Cloud provider and consumer.

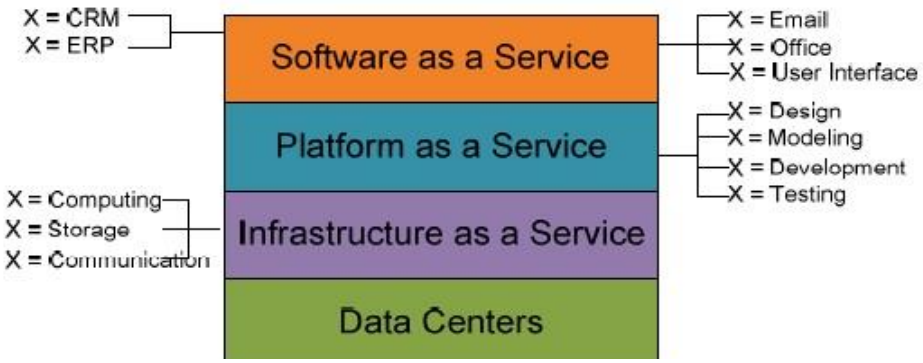

Figure 2.2 Hierarchical View of Cloud Computing [159]

Cloud computing services can be delivered through three types of delivery modes; Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [20].

- *SaaS* is basically delivering a service/application to a costumer utilizing Service Oriented Architecture (SOA). The application is location independent and can be accessed through various devices; either a thin client interface or a program interface. One of the advantages of SaaS is that everything is abstracted and the user would not worry, manage, or control the infrastructure of the Cloud. However, one of the disadvantages

of SaaS is that users have limited configurations settings (control) over the service. One example of SaaS is Google Gmail and salesforce.com.

- *PaaS* enables the costumer to deploy or run their own applications (whether developed/created by them or not) leveraging the Cloud platform, tools, and programming language. Also, managing and controlling the Cloud infrastructure is not the costumers' role. However, they have full configuration settings over the applications, since they own it. One example of PaaS is Google App Engine.
- *IaaS* refers to delivering the computer infrastructure as a service. Consumers are able to deploy their arbitrary software, such as operating systems. They have the power to control and manage all of the fundamental computing resources (i.e. storage, operating system) but with limited control of selected network components (i.e. firewalls). One example of IaaS is Amazon's Simple Storage Service (S3).

The targeted delivery model for this research is IaaS because in which the virtualized and physical resources and deployed, managed and monitored.

2.2.4 Cloud Deployment Modes

The deployment mode of a Cloud data centre depends on the physical location of the computing resources, the organization's responsibilities and the business strategy. There are a number of Cloud computing deployment modes that are being used, which are Private Clouds, Public Clouds. Community Clouds and Hybrid Clouds [23], [24].

- *Private Clouds:* The environments are characteristically customized with devoted virtualized (provisioned) resources and infrastructure for particular organization(s) [28]. The ownership, managements, and

control of the private Clouds by the same organization(s), a third party or both of them. Examples of Private Clouds can be found in enterprises and universities.

- *Public Clouds*: The infrastructure is virtualized to be used by the general public, such as Gmail. The ownership, managements, and control of the Cloud is by a business or an organization (i.e. government or NGO) and the location of the Cloud is in the site of the Cloud provider. Usually services of this category require subscription. Google App Engine, Windows Azure, and Amazon Elastic Compute Clouds are examples of Public Clouds.
- *Community Clouds* are customized for a specific group of consumers who have the same goals or concerns [20], [29]. For example, Siemens offer services for media companies by utilizing Microsoft Azure, a public Cloud, as the underlying Cloud platform while relying on Siemens IT Solutions and Services data centres, a Private Cloud [30].
- *Hybrid Cloud is the* infrastructure composition of two or more unique entities of Clouds (Private, Public, or Community). Those entities remain separate (unique) but are linked together by standardized technologies to enable application and data sharing (portability) [20], [28], [29].

The only accessible type of the deployment modes for this research is Public Clouds, i.e. Google's Dataset. Thus, it has been utilized to meet the objectives of this research.

2.2.5 Virtualization

Virtualization of resources is not unique to Cloud computing, but it is a key unit in Cloud data centres [31], [32]. Virtualization of resources at the lowest level is a feature that differentiate Clouds from Grids [33]. It refers to the abstraction of computing resources, normally as virtual machines (VMs), with related network and storage interconnection [34]. Cloud-powered technology governs the allocation, delivery and presentation of these virtualized resources. With virtualization, rapid and dynamic scaling of resources depending on consumers' demands is allowed [35]. It makes possible for hardware resources to be efficiently used by sharing the same resources to several units at the same time [36]. The Virtualization layer sits between the physical infrastructure (hardware) and the operating system (OS) and the applications. A key element that enables virtualization is the Hypervisor, a Virtual Machine Manager (VMM), which hides the physical resources of the system from the OS. The VMM directly controls the hardware resources, thus allowing the possibility of running more than one OS on the same hardware. As a consequence, the hardware can then be partitioned into logical unites which are called VMs [34]. To the OS and the consumers, a VM appears as an isolated physical machine (PM). Also, the VMM is responsible for the deploying, migration, monitoring and deletion of the VMs.

2.2.5.1 Virtualization Types

There are numerous types of virtualization including Full Virtualization, Para-virtualization and OS-Layer Virtualization [33], [34] illustrated in figure 2.3.

- *Full Virtualization:* The VMM allows complete abstraction from the underlying physical hardware, i.e. CPU and Memory. Both the OS and the

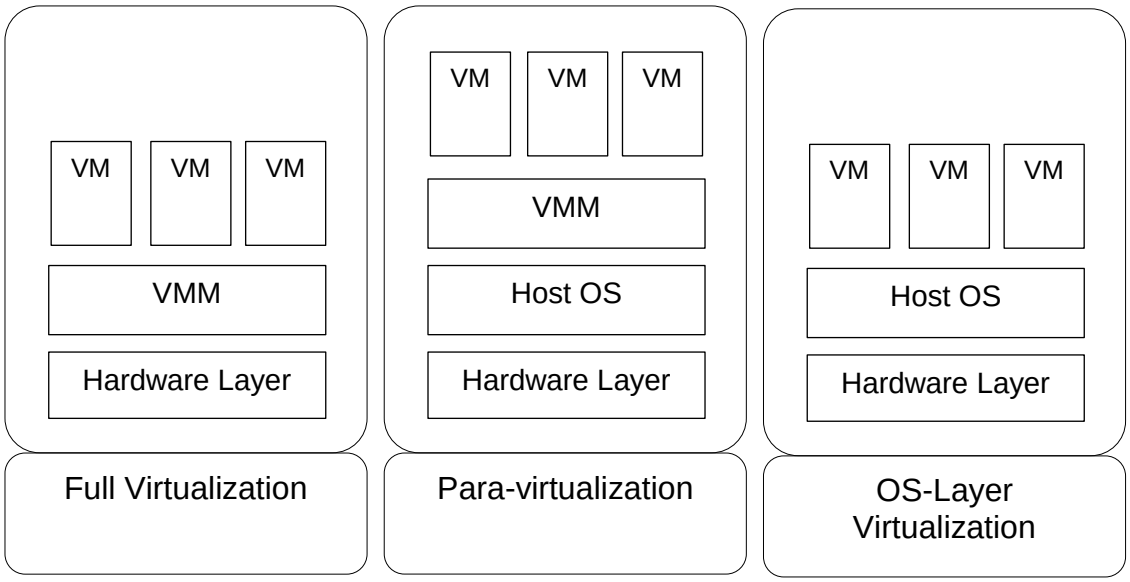

Figure 2.3 Types of Virtualization

VM are unmodified and unware of the virtualization environment. The total abstraction introduces overhead which reduces level of performance.

- *Para-virtualization* is similar to the full virtualization but the OS is modified in the sense that it is aware of the virtualization environment and also aware of the other VMs' resources demands within the same PM.
- *OS-Layer Virtualization:* The VMM runs more than one instance of the same OS in parallel. In this type, both the hardware and the OS are virtualized. The VMs on the same PM use the same OS.
- *Linux Container* is a virtualization tool different from VMs. It allows the ability to package applications and their dependencies into lightweight containers that move easily between different distros, start up quickly and are isolated from each other. A distro is a computer software distribution package. So, a container can be considered as a lightweight equivalent of a VM [37].

2.2.5.2 Virtualization Advantages

According to [34], [38], there are benefits for the organizations who implement system virtualization. The main advantages are described below.

- *Cost.* Resources are accessed and managed efficiently which leads to cost reduction in both hardware and operations.
- *Workload Optimization.* Resources are better utilized by reducing the amount of idle or non-used resources. This allows Cloud providers to maximize the utilization of the available resources. Reacting to high usage, providers can easily shift resources between VMs.
- *IT flexibility.* Creating and deploying VMs can easily be done on demand by Cloud providers regardless of the location. Also, the specification of the VMs can be modified while running such as CPU units and amount of memory.
- *Availability.* In case of PMs taken offline for maintenance, the VMs can keep running. This is allowed by temporarily migrating the VM to another running PM. PMs can be maintained, upgraded and changed without having an effect on the running virtualized instances.

2.2.5.3 Virtualization Disadvantages

- *Single Point of Failure.* The virtualization layer relies on the hardware that hosts the VMs even though it is decoupled from the physical layer. The PM failure can lead to the failure of all hosted VMs which often translates in loss of consumers' data.
- *Overhead*. There is a trade-off between flexibility and performance when applying virtualization. The increased flexibility causes overhead which

has a negative effect on the overall performance of the hosted applications.

2.2.6 Cloud Quality of Service

Guaranteeing a QoS is a pivotal issue for Cloud providers because it determines their level of success [39]. Cloud providers' reputation and revenue depend on the successful delivery of the Cloud services to consumers as expected. A QoS is an assurance that the required resources and parameters would be fully supported, such as levels of performance, availability, privacy, security and dependability [33]. Providers rely on SLAs in order to facilitate and establish the QoS.

A SLA is a legal service contract between a Cloud provider and a Cloud user which includes the QoS parameters that guarantees the quality of service required [24], [33], [40]. It includes the set of services that will be delivered, definitions of each service, the responsibilities of the provider and the consumer, a set of measurable metrics, an auditing framework for monitoring, the consequences and actions for service delivery failure, etc [24], [39], [41]. For example, when a virtualized CPU is under overload, the agreed level of performance is likely not to be met. The same thing for a virtualized memory in a situation of overload, the agreed level of memory is likely not to be met. Also, response time, execution time, latency and more are related performance metrics included in the SLA. There are specific SLA metrics for each Cloud delivery model (explained in Section 2.2.4) [40]. The SLA of IaaS includes metrics such as time for VM to be ready for use, maximum and minimum VMs available for dynamic scaling, availability of access in terms of uptime. For PaaS, SLA metrics include method of charging, maximum number of unique user

access, possibility of integration with other platforms, etc. For SaaS, SLA metrics include reliability, usability in terms of method of access, uptime of software, etc. In addition, SLA coverage can be classified into four levels depending on the delivery models; Facilities-Level, Platform Level, Operating System Level, and Application Level.

2.2.7 Differences between Cloud and Traditional Datacentres

A datacentre is a facility that houses PMs, namely servers, storage units and network devices, systems for power distribution and systems for cooling [42]. Cloud computing refers to the delivery of SaaS, PaaS and IaaS to consumers over a network and the software and hardware in a datacentre that support that delivery [22]. It builds on the principles and paradigms and shares similar characteristics with multiple distributed systems technologies such as Cluster computing and Grid computing [32], [33]. Managing large scale resource

Table 2.1 Differences between Cloud and Traditional Datacentres

Element	**Cloud Datacentres**	**Traditional Datacentres**
Unlimited computing resources on demand	Yes	No
No Up-front obligation by customers	Yes	No
Pay per consumption	Yes	No
Economies of scale	Yes	Usually not
Increase utilization via virtualization of resources	Yes	No
Full control over data and physical equipment	No	Yes
Rapid elasticity of computing resources	Yes	No
High Potential for building value-added or 3rd party solutions	Yes	No

intensive application is the shared motivation between Grids and Clouds. However, Cloud computing leans more towards a business model. A Cloud datacentre is a group of virtualized and connected computers that are dynamically provisioned and provided as combined computing resources which depend on the agreed SLAs [19]. Cloud implementations are deployed over traditional datacentre; thus the approaches at infrastructure level are similar. Method of access to datacentres and the way providers and consumers interact is where Cloud datacentres is different than traditional datacentres. Cloud datacentres are off-site (outside the physical location of the organization) and permit the access of resources through the internet. Consumers are allowed to acquire and reacquire computing resources without the need of human interaction. In comparison, traditional datacentres are mainly on-site (inside the physical location of the organization) and consumers generally need to request resources without guarantee of delivery. Table 2.1 summarizes the differences between Cloud and traditional datacentres [21], [22], [43], [44].

2.2.8 Challenges for Adopting Clouds

Even though Cloud computing offers wide range of benefits, it yet holds many hurdles that disrupt its rapid adoption [21]. Such challenges include business continuity and service availability, multi-tenancy, access control, confidentiality, data integrity, and audit [22], [45], [46]. A number of which will be explained below.

Business continuity and service availability are one of the major challenges in Clouds. In the business community, it is known that there is no guarantee for a company to stay in business for ever. So, there is a risk that, for any reason, Cloud providers go out of business. Also, nearly every major Cloud provider, i.e.

Amazon's S3 and Google's App Engine, has had a downtime in their service availability due to many reasons, such as overload and programming error. One of the suggested solutions to lower the risk of this issue is for Cloud providers to have multiple data centres located in different locations and for Cloud consumer not to rely on one provider (A single point of failure) [22].

While multi-tenancy in Cloud environment introduced many benefits, it brings security and privacy vulnerabilities and threats to both the Cloud users and Cloud infrastructures. In the physical environment, similar functionalities with existing operating systems and applications are pooled in virtualized environments, thus software bugs and recognized security weaknesses in these systems remain the key risk to any virtualized multitenant environment [45].

Having private and sensitive data in off-site Clouds brings about more security threats, such as potential insider attacks or server compromise [45], [46]. Attackers can mimic an identity of a user and potentially be enabled to full access of the data. One of approaches to mitigate this issue in access control is to encrypt the data in a differentiated manner and only share the decryption keys with the authorized users [45]; however, this solution comes at a cost of performance. Trade-offs is almost in every suggested solution of Clouds challenges and issues.

2.3 Provenance

Provenance is not a new idea. Moreau et al gave a simple scenario of provenance in the art world [9]. When a painting is sold to someone, it is often accompanied with a paper trail or physical markings, documenting the details of the first owner (the artist) from its creation to its current state and ownership. This documentary history is referred to as the provenance of the object. The price of

this object depends on the level of completeness and the quality of its provenance, which reflects its importance. In the literature, provenance is sometimes called lineage [47], [48].

Within the context of e-Science, Simmhan et al [49] stated that Data Provenance as:

> *"One kind of metadata which involves recording the dependencies amongst datasets"*.

That dependency shows the relationship amongst these datasets since their creations as well as the information on how they were generated in the first place. In the scientific domain, one benefit of data provenance is enabling the regeneration of missing or deleted datasets, as proposed by Foster [50] in [51] in their notion of virtual data.

Further, provenance is not only about the production of scientific data but also the process that lead to their generation [49]. The granularity at which provenance is collected defines its usefulness. There are two crucial attributes of the provenance of a data item: (1) the ancestral data product from which this data item has evolved, and (2) the process of transformation of this ancestral data product, i.e. workflow, that leads to the derivation of this data products[49], [52].

Buneman et al in [53] defined provenance within the domain of databases as:

> *"The description of the origin of the data and the process(es) by which it arrived to the database".*

This implies that the processes of data transformation and derivation of data should be recorded. So, decision making will be more accurate given data

provenance is in place as a decision mostly involves a human agent who relies on system output. Thus it is pivotal for a user to place trust on the system outputs [54].

Provenance is not new in computing and IT systems as it is being used with regards to the debugging of systems [55]. Provenance data and logs are completely different. Logs provide a time-line history of actions - that have been pre-defined - relating to a single application, whereas provenance data goes beyond logs as it includes data about numerous applications, components and people [55]. Logging and auditing are heavily being used to establish the first point of an error, the reason it took place, other attributes lead to this error being created and the impact on the overall system. It has been claimed that complex systems such as Clouds that include many layers of interactions between software and hardware and involve the interaction between different components from different providers lack such effective debugging systems (logging and auditing) [49], [55]. Thus, it is thought that provenance would bring instant benefits to both users and Cloud providers once it has been introduced to Cloud computing [56].

2.3.1 Provenance in the Web and Cloud Domains

World Wide Web Consortium (W3C) has defined provenance as

> *"A record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing"* [57].

In addition, a more related definition of provenance to this research is introduced by Moreau et al [9] stating that provenance in the context of web and Clouds is defined by

"A record that can be created by, exchanged between, and processed by computers."

For this research, provenance can be defined as *a documentation of Cloud data or descriptions of Cloud events that computers and people can create, exchange, and process*. This documentation records who created what and when, why it has been created, and the path this object has travelled. The 'who' represents the person who initiates an event to be processed. The 'process' can be deploy, update, pause, terminate a task, etc. The 'when' represents the time; for example, the time the event has been created and the time needed for that event to be processed. Events are of five types: invalidation, generation, start and end of activities, and usage of entities [9]. For example, a Cloud activity has a lifetime delimited by its generation and invalidation.

2.3.2 Provenance Models

Because the underpinning of provenance models is commonly a graph representation, it is important to explain it first.

2.3.2.1 Representation of Provenance

It has been stated that the state of an object can be affected by the people involved, the organizations that people act on behalf of, processes executed, and other relevant data [9]. Thus, to better show provenance record in a meaningful way, it can be represented as a graph, that includes data, processes, organizations, and people as nodes and the relations between these nodes as edges [11].

The provenance graph is defines as

"A record of a past or a current execution ... but not a workflow to derive future data." [58].

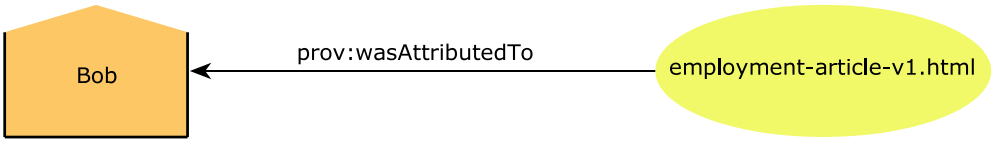

Figure 2.4 The Employment Article was Attributed to Bob. [9]

Capturing the dependencies between entities is the aim of the provenance graph. So, edges that link nodes such as process, artefact, or agent, represent such dependencies between the effect (source) and the cause (destination) [58]. To illustrate, edges can articulate the following reliance: an artefact 'was generated by' a process; a process 'used' an artefact; a process 'was controlled by' an agent; an artefact 'was derived from' another artefact; a process 'was triggered by' another process and so on [9], [58]. Figure 2.4 explains one of the dependencies in an example. It illustrates one angle of the provenance graph, the responsibility angle between an agent, Bob, and an entity, the article. All of which will be described in the next sections.

Other than bespoke models, there two notable provenance models which are widely used, The Open Provenance Model (OPM) and PROV model.

2.3.2.2 The Open Provenance Model

OPM is the outcome of the Provenance Challenge series in 2006 – 2010. The first challenge was to understand the different capabilities of different provenance systems and the second was to utilize provenance information to establish inter-operability of systems. The last challenge was about sensibly evaluating the Open Provenance Model, from an inter-operability standpoint [58]–[60].

OPM is a model explaining artefacts and their derivations in the past; whereas process could be still running or finishing in the future so long as they have originated in the past. Additionally, the OPM is established for the aim to fulfil the following requirements [60]:

- Exchanging provenance information between systems based on a shared provenance model.
- Ability to build and share tools operating on a provenance model.
- To define the model in a precise, technology-agnostic manner.
- Representing the provenance for anything digitally.
- Defining a set of rules and guidelines that specify the valid interpretations that can be made on provenance graphs.

OPM is depended upon three nodes, which are: *Artefact (A)*, a slice of state that may have a physical or digital representation; *Process (P)*, new artefacts resulting from an action or a series of actions executed on or originated by already existing artefacts; and *Agent (Ag)*, circumstantial entity acting as a

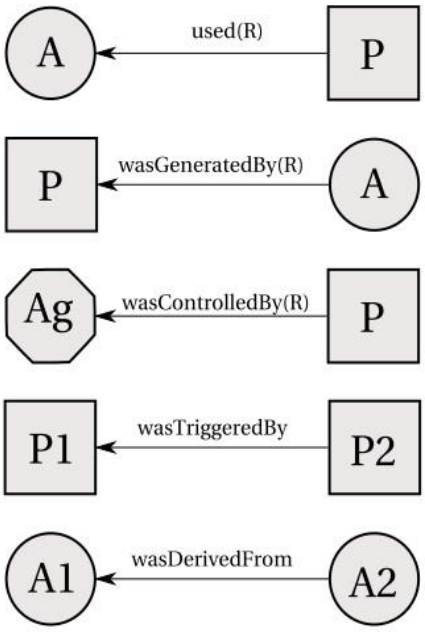

Figure 2.5 OPM edges [58]

medium of a process, simplifying, enabling, controlling, or influencing its execution. With regards to the provenance graph, Figure 2.5 shows the five identified causal dependencies between the three nodes. Circles represent artefacts; octagons indicate agents; and rectangles denote processes. A causal dependency is defined as a relationship that shows the existence of dependency between an effect (source) and its cause (destination) [60].

2.3.2.3 PROV Model

It is vital for provenance to be independent of the technologies used across multiple systems' executions as it is anticipated to express the flow of data and information of those systems. In addition, those systems are probably to be different in nature, employed and designed by different vendors, where each of which may adopt a unique way of representing information. Thus, the PROV model, W3C standard for provenance, implements the notion of the abstract data model PROV-DM, which can be serialized in a number of formats [9]. According to W3C [11],

> *"The PROV Family of Documents defines a model, corresponding serializations and other supporting definitions to enable the inter-operable interchange of provenance information in heterogeneous environments such as the Web."*

PROV family consists of 13 documents which define the factors essential to achieve the task of inter-operable exchange of provenance data and information in different environments such as distributed systems and the web [11]. W3C have recommended 4 documents to serialize PROV model:

- *The PROV ontology (PROV-O)* is an owl2 ontology enabling PROV data model to be mapped to RDF [61], which presents numerous serialization formats, such as rdf/xml.
- *PROV-XML* is an xml schema which allows native xml representations for the PROV data model [62].
- *PROV-N* is a notation for provenance intended for human use.
- *PROV-DM is the PROV data model for provenance.*

PROV model is composed of three classes (provenance views) and seven properties (prefix with ‘prov:’) [9], shown on Figure 2.6.

- The *data flow view* shows transformation of things in physical or digital domains. Also, it is the flow of information within computer systems. Entities (prov:Entity) are arbitrary things or digital artefacts of which we want to describe the provenance. Derivation, encoded by the property prov:wasDerivedFrom, refers to the transformation and the flow of these entities.
- Sometimes, cataloguing the processes that occurred and all related timing information is helpful to offer more information about derivations. The data flow view can then be cleansed by the *process flow view* detailing the activities that took place, in addition to their start and end timings. The property (prov:used) is a notion describing Entities being input to Activities (prov:Activity), which their output are new entities (the notion of Generation described by the property prov:wasGeneratedBy). Additionally, the concept of Communication, articulated by prov:wasInformedBy, captures the flying data from one activity to another.

- Provenance is also about conveying responsibility for what happened in a system, *the responsibility view*. The class of things found in the range of three properties is referred to Agent (class prov:Agent). Agents may be responsible for (a) other agents which forms a Delegation, represented by prov:actedOnBehalfOf (b) for past activities: that is an Association, which is denoted by prov:wasAssociatedWith, or (c) for the existence of entities which refers to Attribution indicated by the property prov:wasAttributedTo.

2.3.3 Benefits of Provenance in the Clouds

Provenance is mainly valuable in difficult circumstances to review complex processes particularly when they involve numerous stakeholders [9]. Provenance is one essential dimension of process verification, reproducibility, reliability and trust in distributed systems [32], [63]. Analysis of provenance information of a given task would pave the way to extract knowledge from usage

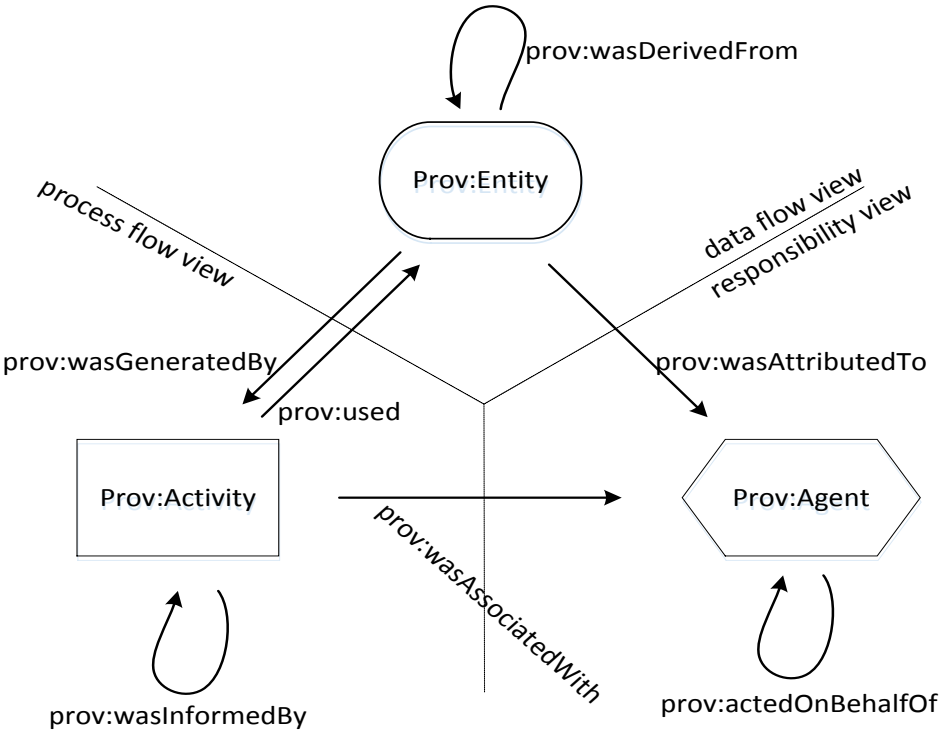

Figure 2.6 PROV Abstract Model [9]

data that was not identified using the standard logging system. According to [49] [64], there are several uses (benefits) that provenance systems could support, some of which are:

- *Attribution* in which provenance can establish the copyright and ownership of data.
- Provenance can be used to assess *Data Quality and Reliability* with regards to the origin data and transformations.
- The more detailed the provenance the more confidence in *Replication* of data derivation.
- Provenance can provide the ability to track the *Audit Trail* of data.

In addition, provenance in scientific and intensive computing is claimed to provide guarantee in quality of results and to assure the repeatability of experiments [55]. Nevertheless, there is one crucial trade-off when utilizing provenance which is scalability. Having more and more data to be recorded adds an overhead on scalability [65]. To guarantee whether data on the web/Cloud is creditable or not takes a lot of effort, cost and time. Clouds provide the feasibility for resources and data to be shared easily (and sometimes anonymously); however, there is no 100% guarantee of the reliability of the data which brings about doubtfulness between users. Thus, provenance is important for Clouds because without it, users will not be able to validate the data's identity and authenticity [66]. In Clouds, location of the physical machine is unknown which makes the normal forensic activities redundant, such as traditional capture and seizure. Data Provenance is one way that can fulfil the need to track Cloud data and users which makes it a key in Cloud forensics field [67].

2.4 Summary

This chapter has provided a detailed summary about Cloud computing. It presented an overview on Cloud computing deployment modes and delivery models and the challenges in adopting the Clouds. It also discussed in detail the Virtualization feature which is an essential key component in Cloud computing. Besides, the benefits of virtualization to both the consumer and providers has been presented.

Furthermore, the concept of provenance and the two standards of provenance models have been discussed. Then, the potential advantages of provenance have been presented

Chapter 3

Literature Review

3.1 Introduction

This chapter presents related work in Cloud resource utilization and provenance. The advantages and disadvantages of resources over-commitment are discussed. Overload issues and the existing mitigating strategies in the literature are presented. The challenges of provenance adoption in the Clouds is presented and discussed. Finally, the Chapter concludes by discussing the related work with regards to the use of W3C PROV provenance data model in Cloud computing environments highlighting the research opportunities based on the gaps found in the literature review.

3.2 Cloud Resource Utilization

This section will explain the relationship between overestimation of resources by users, over-commitment of resources by providers, and the inevitable overloading in physical machines if over-commitment is not administrated carefully. It also discusses the existing overload reactive mitigating strategies.

3.2.1 Overestimation of Resources

As described in Section 2.2.1, *on-demand self-service* is one of the characteristics of Cloud computing. This feature allows resources to be requested, managed and used by consumers without providers' intervention. For example, consumers acquire substantial virtual compute resources such as CPU to enable the configuration and deployment of their applications and platforms on the physical infrastructure. It is evident that consumers lack the understanding and knowledge to determine the exact needed resources to compute their tasks, thus they tend to estimate and consequently pay more than for resources than

they require [68], [69]. According to [70], [71], consumers tend to significantly overestimate the required Cloud resources as there is a large difference between the estimated assigned value and the actual usage consumption of that resource by a task.

From the Cloud data centres' perspective and complying to the feature of *on-demand self-service,* all requested resources by consumers are granted virtually. This may lead to the issue of physical resources underutilization due to the fact that hosted tasks on VMs rarely reach the peak demand simultaneously [1], [72]. Another scenario for underutilization of resources is caused by the fixed VM sizes offered by providers [73]. Underutilized/idle PMs consume considerable power in data centres, almost 50%, which leads to losses in revenue [1], [74]. Therefore, Cloud providers apply over-commitment policy in order to maximize profits, utilize resources and mitigate the problem of overestimation [2], [75], [76].

In order to provide an illustration of the overestimation problem, an analysis has been done on a real Cloud usage dataset by [71]. Figure 3.1 shows the hourly average of CPU (top) and memory (bottom), actual usage (left) and allocated

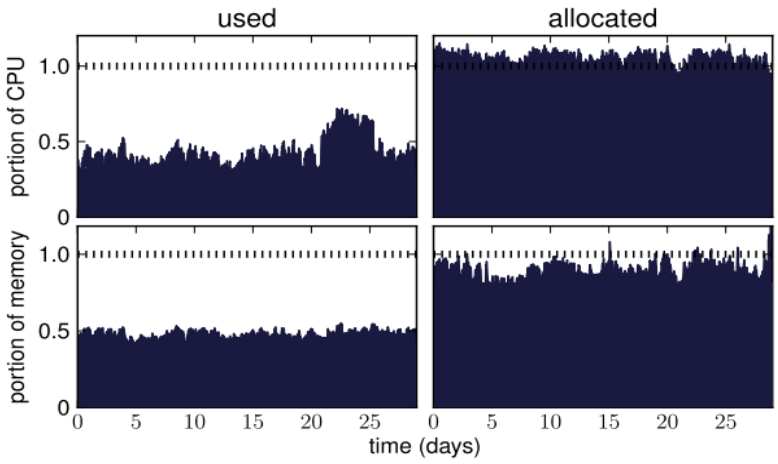

Figure 3.1 CPU and Memory overestimation and over-commitment observed in Google Cloud dataset [71]

resource (right). The total capacity of the data centre is represented by the dashed line seen on the top of every plot. According to [71], the dataset shows that the data centre is heavily booked. As it can be observed, the total resource allocation is more than 80% of the data centre's memory capacity and more than 100% of the data centre's CPU capacity at almost any time. However, the actual demand is much lower. On average, the actual memory usage and CPU usage is about 50% and 60% of the actual physical capacity, respectively. In addition, CPU has been almost always been overcommitted whereas Memory has not been as much overcommitted. The Google usage dataset will be discussed further in Chapters 4 and 5.

3.2.2 Over-Commitment of Resources

Over-commitment of resources, also known as oversubscription, over-allocation or overbooking, is the practice of allocating more virtual resources on a PM than the actual physical capacities [74], [77]. Every Cloud provider has specific policies to determine the amount of Over-Commitment Ratio (OCR) because higher ratios introduces higher risks [1], [73]. OCR is set by the provider which is the highest limit of the physical resources' over-commitment . For example, if the ratio is 2:1, then a PM with a capacity of 50 CPU units can accept requests of 100 virtual CPU (VCPU) units. According to [1], [78], a larger OCR can increase the density of the workload per PM but negatively impacts on the performance guarantees. A low OCR impacts positively on performance but does not greatly increase levels of resource utilization. When determining the OCR, Cloud providers need an understanding of the type and characteristics of the tasks and jobs to be executed and be aware of the agreed SLAs and the data centre's capacity and infrastructure [78]. It can be argued that OCR can be adjusted dynamically in order not to experience overload in the machines. It can be done

by setting a maximum usage level by which OCR is changed once the level is reached or exceeded. Adjusting OCR dynamically might prevent future occurrences of overload; however, overload can still occur due to running tasks prior to OCR adjustment in addition to other causes irrelevant to OCR, as discussed in section 3.2.4. In addition, OCR only works for new tasks. This research focuses on submitted and running tasks. The underlying assumption of over-commitment is that the allocated capacities will never reach total consumption at the same time. Providers utilize such practices in a leverage to serve the same number of consumers with less PMs and to mitigate the problem of overestimation by users [1]. Over-commitment not only increases the average utilization of a data centre or a cluster but also increases the number of tasks and jobs that can be supported and executed [78], [79]. Although over-commitment is beneficial, it introduces risks such as PM overload [4], [80], [81].

3.2.3 Machine Overload and Mitigating Strategies

As explained in Section 1.1, the ideal expectation is when the actual physical capacity is more than or equal to the actual usage and less than the requested capacity ($Ract <= C(R) < Rreq$). However, the case when the actual usage is higher than the actual physical capacity is called overload ($Ract > C(R)$). In the case of overload, new requests can no longer be accepted or fulfilled due to the limited physical capacity available [43]. According to [73], [74], [82], [83], PM overload can lead to VM disruptions and performance degradation in terms of low latency and response time. It also causes resource shortages [81]. Overload occurs when the actual usage demand exceeds the physical capacities [84]. Overloads can happen to all types of resources, namely CPU, Memory, Disk, and Network bandwidth. Not managing overloads can lead to violations and breaches to the SLAs, described in Section 2.2.6.

Cloud service providers face three different types of costs as a consequence of overloading. Repair costs when overloading lead to components failure, Penalty costs due to services disruptions, and Business revenue losses due to constant services outages and unavailability [85].

There are different strategies to mitigate PM overloading, namely Resource Stealing, Quiescing, Live Migration, Streaming Disks, Network Memory, and Task Eviction [2], [4], [7], [84], [86], [87].

- *Resource Stealing* refers to the idea of downsizing the underloaded VMs or PMs and putting the free resources into the overloaded VMs or PM. In distributed systems, resource balancing facilitated by resource stealing is a crucial step especially if it's achieved dynamically which helps maintain high overall system utilization. The controller checks available resources before hosting a task. In the case of lack of resources, the controller allocates the local available resources, if any, then queries other VMs or partitions to steal from them the remaining needed recourses [88].
- *Quiescing* means turning off a number of VMs to regain balance in the PM and then they can be resumed. In an overloaded PM, one or more less significant VMs get quiesced so that the remaining more important VMs run normally and don't face any performance degradations. Once the running VMs complete their tasks and resources become free, the quiesced are resumed [89].
- *Live Migration* means dynamically migrating an overloaded VM (or an underloaded VM from an overloaded PM) to another PM that has enough space to host it. The common approach for live migration of virtual machine is through pre-copy. On the destination host, a shadow VM is

created. Each used memory page is then copied from the source to the destination. While the memory pages are being copied, the VM is still running on the source host until the before coping the last memory page. The VM stops on the source host and resumes on the destination host [90], [91].

- Streaming Disks means migrating enough portions of VM's local disk from an overloaded PM to an underloaded PM so that it enables the VM to start. Once load is balanced, the remaining portions of the VM are transferred. This strategy reduces network costs of migration to the disk [84].
- *Network Memory* enables the provider to make use of a memory from another machine over the network. Until load is restored, page repositories across the network are used for VM swap pages [2]. Swapping can potentially ease load on the local disk of the overloaded PM [4].
- *Task Eviction* is where a hosted task on a VM is pushed out of the VM to re-queue (re-submit) and wait for free resources to be hosted again. Tasks sometimes are no longer fit to run on the machines and get evicted due to (i) hardware failure, (ii) overload because of machine's over-commitment, or (iii) competing workload [71]. There are policies that are used to determine which tasks to be evicted, such as task's priority. Most evictions are resultant to changes of machine configuration or other higher priority tasks being hosted and started on the same VM [71].

These strategies are mainly reactive. They are triggered after the overload has taken place. Selecting a specific mitigating strategy by Cloud providers depends

on the characteristics of the data centre and the running applications [2]. Possible proactive methods such as FTM, TCloud, FTCloud are discussed in section 3.2.5. However, to the best of our knowledge there are no proactive methods found in the literature that directly targets the issue of overload mitigation and prevention.

3.2.4 Overload Causes

Machine overload can be caused by variety of reasons in addition to over-commitment. Both the user and the provider can contribute to the existence of overload [2]. For example, increasing the request of resources by user can lead to overload in the case that not enough space is available. Bad management of resources by the provider could lead to memory leaks, for example, which also can lead to overload. Also, increasing the over-commitment ratio by providers could increase the chances of overload occurrences [73]. In addition, physical resources could cause overloads. For example, fan failures and CPU overlocking could lead to CPU overheating and server overload [92]. Google have observed five different causes of overload in their data centres; namely Take Over by Higher Priority Tasks, Increase In Resource Requests, Demand Exceeds Physical Capacities, Missing Machines, and Decrease in Machines Capacities (refer to sections 4.3 and 4.8) [7], [71]. Overloads can be in three different states; (i) network overload (i.e. bandwidth), (ii) hardware overload (i.e. PM), or (iii) software overload (i.e. VM) [93], [94].

As explained earlier, overload is generally caused due to the limited capacity of the free physical resource. In practice, overload can be caused due to a combination of more than one cause in addition to the limited physical capacity because of over-commitment, refer to section 5.5.

There are several causes that may lead to overload which can later be mitigated by six different strategies. The scope of this research is to investigate one mitigating strategy and five causes that triggered it using a real Cloud usage data for learning and exploration, refer to chapter 3. The mitigating strategy is Task Eviction and the causes are: Take Over by Higher Priority Tasks, Increase in Resource Requests, Demand Exceeds Physical Capacities, Missing Machines, and Decrease in Machines Capacities, refer to chapters 4 and 5.

3.2.5 Causes Diagnosis and Identification

In distributed systems, machine overloading is an issue. There are existing concepts which can be utilised to diagnose overloading such as fault tolerance, self-healing, Data Centre Infrastructure Management (DCIM), and rule based techniques. These methods are considered as resiliency techniques in Cloud computing [85]. In this section, these concepts and tools, such as TCloud, FTCloud, Chopstic, Fay, D3S, and Pip, that could potentially be utilized for overload causes identification will be presented.

In [95], authors stated that fault diagnosis is valuable as it helps administrators and developers to identify causes of disruption. It has two limitations. First, it is always passive and reactive. The diagnosis is triggered after the disruption has already occurred and the symptoms have appeared. Second, diagnosis only detects the issue but does not prevent it. Disruption diagnosis, nevertheless, is a prerequisite step before undertaking any kind of corrective measures and it is a crucial indicator of systems resilience [96]. In addition, employing only fault prevention techniques, such as rigorous development process, and fault removal techniques, such as debugging, have been proven to be difficult [97]. Increasing the dependability of distributed systems and critical applications can be achieved through the use of fault tolerant methods [98]. Implementing fault tolerance in

Cloud computing is still a challenge [99]–[102]. Challenges range from the heterogeneity of the integrated components from different vendors to the difficulty of integrating existing scheduling algorithms with fault tolerance approaches. The role of fault tolerance has been described in [103] as:

> *"… to preserve the delivery of expected services despite the presence of fault-caused errors within the system itself. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service."*

Failures, errors and faults are three different things [103]. When the system deviates from its pre-specified behaviour, a failure could occur. Failures are caused by system errors. An error represents an invalid system state. An error is a result of a fault, which is a defect in the system. Thus, a fault is the root cause of a failure. Examples of failures in distributed systems are hardware faults, physical machine overload, and network congestion. Fault tolerance techniques are either reactive or proactive [104]. In reactive methods, the fault's impact is reduced using recovery methods but only after it has occurred. In proactive methods, faults are predicted, then fault occurrences are prevented. The latter is preferred but there is still no guarantees that predictions are always accurate [104].

There are three well-known fault tolerance approaches [97], [98], [103]. *Recovery Block* approach is a mechanism implemented in software fault tolerance where redundant program modules (components) are structured. Standby components are sequentially invoked if the primary component fails. *Multi-Version Design* approach is a mechanism that generates multiple functionally equivalent versions of a system. When applying the Multi-Version

Design approach to Cloud services, the functionally equivalent versions are invoked at the same time and their results are compared. The final system result is determined by the consensus output. *Parallel* approach is similar to Multi-Version Design. The difference is that the final result is the first returned output.

In [105], [106], authors propose a Failure Tolerance Manager (FTM) in the Clouds which applies the tight integration of management components technique [85]. FTM addresses the issues of different computing resources and integrates three managements components to realize generic fault tolerance mechanisms and to enable adaptable resilience in Cloud environments: (i) *Resource Manager* avoids failure and congestion when allocating resources and manages the network links, (ii) *Replication Manager* supports the replication mechanisms which is used for fault healing. It manages their execution by ensuring that fault-free replicas show correct behaviour during execution, (iii) *Fault Masking and Recovery Manager* enforces fault masking mechanisms to prevent error occurrences and ensures that the detected system faults are hidden from the user, such as VM and node faults. The recovery manager's goal is to minimize the downtime of the system during failures by recovering and resuming the last error-free replica of the system. Fault tolerance techniques can be applied to an overload scenario only when an overload cause is considered as a fault, an error or a failure such as virtual machines failures. Due to the characteristics of resources over-commitment and machines overload, there are causes that don't fit into these three categories, such as arrival of higher priority tasks and increase in resource requests. Although fault tolerance techniques are effective, they are not fully applicable to the different overload scenarios. The proposed provenance-driven diagnostic framework contributes by diagnosing every overload scenario and identifies the linked causes, refer to Chapter 3.

Self-healing is another resiliency technique that is applicable to the Clouds [85], [96]. Dai et al [107] propose a self-diagnosis and self-healing tool that utilises both Multivariate Decision Diagrams and Naïve Bayes Classifiers as a hybrid mechanism. Self-healing is the ability of the system to autonomously discover, diagnose and mitigate any disruptions in the system. Their proposed tool is based on a consequence-oriented concept. The tool diagnoses the detected symptoms to prevent failures. The Multivariate Decision Diagram determines the severity levels of the possible failure. The Naïve Bayes Classifier is a probabilistic classifier and it infers the possible consequences. The tool scope is in both hardware and software as it looks, for example, for the causes of memory leaks and programming bugs. Although authors claim that this is a self-healing tool, it goes as far as prevention. The tool can be extended and utilized to be used as an implementation of the overload mitigating strategies and to help decide which the best strategy is by utilizing the Naïve Bayes Classifier. Verissimo et al [108] proposes a failure preventative tool, TCloud, which its goal is to reallocate resources and reassigns trust in in hardware and software components. TCloud aims to detect and prevent failures. TCloud has been extended to FTCloud which adds the capability of fault tolerance [97]. FTCloud uses a ranking approach for resources allocation. Each Cloud component is tested and ranked in terms of availability. Higher-ranked components are used for the allocation. A drawback to TCloud and FTCloud is that a data centre needs to be re-implemented and reconfigured in order to be compliant. Following the formulation steps of the proposed framework, section 4.4, it can be seen that reconfiguration of a data centre is not a prerequisite. Kavulya et al [96] argue that self-healing is risky because it depends on the outcome of the diagnosis. Wrong diagnosis could

lead to bad and wrong recovery decisions. This adds importance and creditability to the diagnosis models.

DCIM is a one of the management solutions for data centres and can facilitate resilience techniques [109]. It is the management layer of the physical infrastructure. It allows data centres to leverage existing technologies such as data collectors, meters and sensors to support capacity planning and analytics and data management, integration and reporting [110]. DCIM can be used as a method for the identification of the overload causes. For example, the decision support system in the DCIM that can be used by the scheduler to identify weak nodes by offering a ranking method of servers. Ouyang et al [111] propose a node performance modelling and ranking framework which analyses node execution ability and vulnerability to straggler occurrence by using Google Cloud 29-day production data as a case study. They stated that the framework could inform the scheduler about node-level stragglers that show weak performance to be avoided. It can observed from section 3.2.4 that the scope of this framework does not cover all causes of overload. This work shows that DCIMs can be further improved.

Diagnosis techniques and methods such as count-and-threshold techniques and rule-based techniques, have historically been a manual process. The notion of automating as much of the diagnosis process as possible has grown in importance. Diagnosis techniques can be used to guide cause analysis and are found in diverse domains such as artificial intelligence, distributed systems, machine learning, statistics, and stochastic modelling. Such techniques are not perfect and can fail to pick up a problem resulting in a false negative (FN) or accuse the wrong and irrelevant cause resulting in a false positive (FP), hence

the importance of tuning [96]. Four existing non-provenance diagnosis tools will be presented next; Chopstic, Fay, D3S, and Pip.

Chopstix is a manual lightweight monitoring tool that collects data of low-level operating system events and analysis them offline [112]. The study claims that Chopstix monitors hardware-related vital signs, such as CPU utilization, then detects and isolates the root cause of a fault, such as OS operations, to allow the use of existing debugging tools by developers. This diagnostic tool is rule-based and relies on a small collection of rules to guide the diagnosis. According to [96], this tool only diagnoses problems on single node and does not correlate data from several nodes.

Fay is a similar rule-based diagnostic and monitoring tool [113]. Collection, processing, and analysis of software and hardware execution traces are facilitated by Fay [114]. It is applicable to activities of users and nodes. It allows the traceability in distributed systems by their developed FayLINQ language, which provides the means to specify the events to be traced, aggregated, processed, and analysed. Rule-based techniques lack (i) the ability to learn from experience and (ii) the ability of dealing with failures not described within the predefined rules [96].

D3S is a large-scale real-time debugging and checking tool [115]. Programmatic tests of measured performance data are used to imperatively describe performance expectations. When a problem is detected, a sequence of state changes that led to the problem is produced to be further analysed by a developer. The drawback of this tool is its complete dependence on the developer to write and specify predicates. Predicates, similar to rules, are injected by the tool into a running process and used as guidance.

Pip is another debugging tool which aids the developer to explore expected and unexpected system behaviour [116]. Similar to D3S, Pip depends on the developer to express, programmatically, system expectations. Pip uses such inputs to compare the actual and expected behaviour to flag any application-level issues or performance problems. Properties of a program, such as throughput, node failure, and latency, can be dynamically monitored and checked by Pip. However, such tools requires a deeper understanding of the systems as a prerequisite [96].

3.3 Provenance

This section presents a review of related work about the application and the use of provenance in the Clouds. It also presents the possible provenance-based methods for overload causes identification.

3.3.1 Challenges of Adopting Provenance in the Clouds

In [32], [55], [67], [117], the authors have identified the challenges and hurdles that face adopting provenance in the Clouds. They have stated that for Clouds, supporting scientific computing as well as fulfilling the common requirements for improved auditing, debugging, and resource monitoring is envisioned to be possible by creating practical provenance systems. As mentioned earlier in Section 2.2.4, there are a number of deployments of Clouds; private, public, and community. The differences in these deployments are, and not inclusive of, number of users, relationships between users and Cloud providers, supported systems and devices, and adopted business models. As a consequence, such differences have a direct effect on the level of urgency to adopt and implement a provenance system. The challenges can be classified into five categories:

1. *Audit and Log Data.* For a Cloud provider, services are of three types, SaaS, PaaS, and IaaS for different consumers. Cloud consumers may require Cloud services from different providers. The current way of recording and collecting provenance in Clouds is by log files [118]. Thus, there is a log for every Cloud deployment mode of every Cloud model for every service at every logging interval which makes it challenging and almost not practical to collect provenance data from logs of services that are of dynamic nature.

   The challenge faced in this research is mapping the data collected through current logging techniques to the provenance model PROV.

2. *Heterogeneity*. Clouds in nature are complex systems and involve numerous interlinked resources that are managed by different policies and offered by heterogeneous Cloud providers. Thus, when investigating an event for one application data from many sources might be needed, which increases the cost of time and effort doing it because an investigation involves the application itself, all logs for the virtual resources the applications used, and the logs for the physical resources that supported the hosting of those virtual ones.

   In terms of this research, only one log of one Cloud provider has been used. This particular challenge has not been faced. However, it is a possible obstacle when extending the proposed framework in future work to include datasets of different format and structure from data centres with heterogeneous infrastructure design.

3. *Granularity*. Scalability is a challenge facing provenance adoption in Cloud computing. According to the literature, there are trade-offs between the granularity of provenance and Clouds' scalability and performance

because recording every single aspect leads to gigabytes of data, important and not important, being created [46]. So, the challenge is to maintain the levels of scalability and performance while refining the granularity of provenance data in terms of scale and level of detail.

The overwhelming volume of the dataset used in this research made it almost impossible to deal with. A number of programming languages such as Python and database platforms such as MySQL Workbench have been used and yet failed to return results in reasonable time. Thus, the level of granularity has been changed in terms of selecting only the needed attributes in order to be queried.

4. *Security and Trust.* For provenance in the Clouds to be trustworthy, it must meet four requirements; Confidentiality, Integrity, Availability and Reliability [67]. Collecting provenance at the kernel-level is one of the ways to determine its security and reliability because generally users have no control and access at this level. Muniswamy-Reddy et al stated there is no clear means of recording provenance in the Cloud [66]. They have identified four features that are crucial for any provenance system to be creditable, which are Provenance Data-Coupling, Multi-Object Causal Ordering, Data-Independent Persistence, and Efficient Query.
5. *Persistence.* Each data object in a Cloud requires an identification. When such an object travels between different Cloud providers, the identification changes due to the different policies followed by the providers. Data objects in the Clouds are transient, thus hashing identification techniques are not appropriate because the hash identifier need to continuously be recalculated which affects the overall performance. The provenance

graph could be disconnected if one provenance object is removed, hence provenance persistence is important.

3.3.2 Provenance Research Projects in Distributed Systems

SPADE [119] is a software infrastructure for collecting and managing provenance data in distributed systems based on OPM which supports provenance auditing in distributed environments. It supports both graph and relational databases for storing data and provides a distributed module for querying. The core of SPADE is a provenance kernel which decouples the gathering, storing, and querying of provenance data from different provenance sources, such as application and operating systems. The kernel consists of four components; *reporter* which collects provenance data, *filter* which undertakes transformation on provenance events, *storage* which is used to store filtered provenance data and *sketch* which is used to optimize the querying process. SPADE utilizes OPM's nodes and edges to embed the domain-specific semantics of the provenance.

STRAPP [10], [120], [121] is a framework that improves trust and the understanding of risks in distributed systems using personalized provenance reasoning and risk assessments techniques. W3C PROV provenance data model has been used to systematically model system provenance. STRAPP consists of three main components: the *Presentation Service* which takes input from the user from an external unit, formats it, then passes it to an internal unit. It is responsible for displaying the final view of the STRAPP system in XML to the user, the *Personalization Service* invokes the provenance model and its reasoning engine, personalizes the provenance view and conducts risk assessments, and *Data Management Service* which retrieves data for the

personalization service. The authors justify their choice of W3C PROV model rather than an ad-hoc approach because it provides a standardized layer of normalized data upon which processing algorithms can be implemented. They stated that created provenance graphs can be modified, parsed and serialized leveraging standard tools. Also, reasoning engines can be invoked for the detection of inconsistencies and missing provenance data, such as agents. It provides the ability to understand relationships between entities without data being present in the underlying database.

3.3.3 Use of Provenance and PROV in the Clouds

A number of studies have considered using provenance in the Clouds for different purposes [55], [56], [66], [122]–[125]. However, there were no attempts of using the PROV model standard that enables the exchange of the provenance information [11]. These studies have developed bespoke models as the PROV model was still being developed. Using standard models can help work undertaken by both research and industry communities to be easily understood and extended by building on them.

PROV model has recently started to gain attention in the Cloud computing community. In [48], researchers applied PROV model in the Cloud for security and trustworthiness purposes. One algorithm has been developed based on PROV model for controlling access to Cloud data. It ensures the completeness of the causal dependencies between the data. Another study used PROV model as a basis for a provenance framework for gathering and storing Cloud workflow provenance data for later analysis [126]. In [127], the study argues that current provenance models lack the express-ability to describe the low-level working of a Cloud service. cProv which is a provenance traceability model and cProvl

which is a provenance-aware policy language are proposed to support accountability and provenance traceability in the Clouds. PROV model is used and extended by cProv to provide a representation of provenance history for the Clouds. The use of PROV notation is to add the missing and need express-ability (relation, metadata) on the Cloud provenance data.

Li and Boucelma [128], [129] used the open provenance model (OPM) and Coloured Petri Net (CPN) for monitoring workflow and data provenance in the Cloud by utilizing SOA. Their approach is similar to the approach conducted in this research, described in Chapter 3. They have used the simulation tool CPNTools to act as the diagnoser for their analysis. CPN is used as the abstract model underpinning the diagnosis component which identifies the correct and faulty behaviours of the workflow, starting from the symptoms (faulty data or activities), and backward detecting the possible causes of the symptoms. Web Service Security (WS-S) protocol has been modelled using OPM to integrate the secure communications in Cloud environments. The reason why WS-S has been chosen is due to its support to provenance requirements and it does not disrupt the generality of OPM model.

In [130], Distributed Time-aware Provenance (DTaP) is proposed which helps in debugging and forensics in distributed systems, i.e. Clouds. The study argues that distributed systems problems in performance, security, or configuration don't always have a sole cause but could have a combination of causes (behaviours). DTaP collects distribution, time, and causality of updates. It also gives the administration control to make ad-hoc queries over network communication patterns, system states, etc. This tool facilitates the manual diagnosis by developers The focus is maintaining the network overhead of provenance

collection in distributed systems. It also discusses that provenance can be maintained and provenance trees could be constructed in two ways; proactive and reactive.

In [122], [131], Provenance Aware Storage Systems (PASS) is proposed by Muniswamy-Reddy et al and Barillari et al. It is a scheme for automatic maintenance and collection of data provenance in Cloud storage systems. Its architecture integrates provenance from several layers of abstraction; hence distributed systems.

In [132], the authors stated that provenance data in the Clouds can be classified into five granularities; application, virtual machine, physical machine, Cloud, and Internet (Cloud of Clouds). Provenance of VM summarizes all provenance data related to a VM. Provenance of PM summarizes all provenance data to a particular PM and the mapping of VMs to PMs. Provenance of Cloud incudes the provenance data across the three layers; application, virtual, and physical and the communication between them. It also includes data such as consumer details, migration of data across VMs, migration of VMs across PMs, and more.

S2Logger in [133] is a data event logging tool which captures, analyses and visualizes Cloud data provenance. This tool enables the near real-time detection of security violations and allows for end-to-end tracing of data events at both block and file level. The scope of this tool is detecting data loss and leakage in physical and virtual machines and supports diagnosis of security breaches. S2Logger builds on and complements existing distributed security systems, such as SELinux. Cloud data provenance related to hardware, software and network is monitored as a graph by S2Logger. The study claims that analysing the data flow graph can help make better Cloud data security decisions.

In [134], a provenance-driven auditing framework is proposed by Meera et al. This framework allow providers to run audit checks to ensure there are no inconsistencies in terms of data, SLAs, etc. It follows the procedures and phases of the digital forensic investigation; acquisition, preservation, analysis, and presentation. The framework is based on OPM. This study aims to provide a secure way of provenance audit in the Clouds using existing cryptographic techniques.

Even though these studies are notable, their aims and objectives are different than the ones of this research. They do not look into the overload problem of physical machines which is the scope of this research.

Table 3.1 summarizes the comparison between the possible provenance and non-provenance based methods, presented in sections 3.2.5 and 3.3.3, that can potentially be used for the identification and diagnosis of overload causes. The comparison is drawn between the capability and the scope. The capability is based on six different metrics: prediction, detection, diagnosis, mitigation, prevention, and healing. The scope shows the coverage and focus of the method in terms of three metrics: hardware (i.e. CPU and Memory utilization), software (i.e. virtualization-related issues and programming bugs) and network (i.e. bandwidth and response time). “Yes” means that the method either supports the metric or paves the way to it. “No” means that the method does not support the metric.

Table 3.1 Comparison of Different Possible Methods for Overload Causes Identification

Method (page in this thesis)	Capability						Scope		
	Prediction	Detection	Diagnosis	Mitigation	Prevention	Healing	Software	Hardware	Network
FTM by Jhawar et al [105], [106] / (p. 43) non-provenance based	No	Yes	No	No	Yes	Yes	Yes	Yes	Yes
Dai et al [107] / (p. 44) non-provenance based	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No
TCloud by Verissimo et al FTCloud by Zheng et al [97], [108] / (p. 44) non-provenance based	No	Yes	No	No	Yes	No	Yes	Yes	No
Chopstix by Bhatia et al [112] / (p. 46) non-provenance based	No	Yes	Yes	No	No	No	Yes	Yes	No
Fay by Erlingsson et al [113] / (p. 46) non-provenance based	No	No	Yes	No	No	No	Yes	Yes	No
D3S by Liu et al [115] / (p. 46) non-provenance based	Yes	Yes	Yes	No	No	No	Yes	No	No
Pip by Reynolds et al [116] / (p. 47) non-provenance based	No	Yes	Yes	No	No	No	Yes	No	No
CPN by Li et al [129] / (p. 52) Provenance based	No	No	Yes	No	No	No	Yes	No	Yes
DTaP by Zhou et al [130] / (p. 52) Provenance based	No	No	Yes	No	No	No	No	No	Yes
S2Logger by Suen et al [133] / (p. 53) Provenance based	No	Yes	Yes	No	No	No	Yes	Yes	Yes

3.4 Summary

The concepts of over-commitment and overload have been discussed. Also, this chapter presented the overload mitigating strategies, overload causes and symptoms and existing possible methods for causes diagnosis and identification. The challenges and issues in adopting provenance in the Clouds along with a number of research projects that utilized provenance and PROV in the Clouds have been presented.

Lastly, the Chapter provided a literature review of the uses of provenance and the PROV model in the Clouds. The review of the literature reveals that PROV model is adoptable in the Clouds but it has never been utilized with regards to the issue of Task Eviction which is an opportunity captured by this research. It ends with comparison of different possible methods for overload causes identification, provenance based and non-provenance based.

Chapter 4

Provenance-Driven Diagnostic Framework

4.1 Introduction

Infrastructure as a Service (IaaS) in Cloud computing has introduced many new opportunities for businesses and individuals by extending accessibility and minimizing costs by providing users with access to remote resources [135]. However, as the Cloud computing paradigm rapidly evolves, the management of resource allocation becomes increasingly important so as to maintain a high level of overall system utilization. These challenges are typically addressed through the use of virtualization and the over-commitment of resources to users.

This chapter proposes the Provenance-Driven Diagnostic Framework which addresses the negative impact due to over-commitment that leads to task eviction, as discussed in Chapter 2. The framework investigates cause and effect relationships. The chapter explains the underpinning concepts of the framework. It presents the importance of provenance to the Clouds and how PROV can help Cloud data centres understand why tasks were evicted. A publicly available Cloud dataset with known Task Eviction behaviours was used to inform the construction of the PROV-TE provenance model. Also, the methodology of building the framework and developing the diagnostic algorithms is described. Finally, an instantiation of the framework, the Auditor, is presented. Specifically, how the framework fits in a Cloud data centre is described.

4.2 Underpinning Philosophy and Assumptions of the Framework

As described in Chapter 2, the record of an activity that leads to a piece of data is the provenance of that data [98]. Provenance describes the flow of data and processes across several heterogeneous layers and systems. The reason for using provenance is because (i) traceability of results is provided; (ii) reproducibility is possible; and (iii) the integration of diverse data sources is facilitated by the schema. Analysis of provenance information of a given task would pave the way to extract knowledge from usage data that was not identified using the standard logging system.

PROV is W3C standard for provenance. As defined by W3C, “provenance is a record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing” [136]. With regards to distributed systems, Moreau and Groth in [9], [137] stated that provenance can relate to data, documents and resources since it is a record that computers have produced, processed, and exchanged. In addition, provenance is one essential dimension of process verification, reproducibility, reliability and trust in distributed systems [63]. PROV, explained in Chapter 2, is a model that represents all types of tangible and intangible objects such as data and machines, and allows the expression of causal relationships and dependencies between them through nodes and edges. The dependencies define the link between the effects and the cause in a backwards manner.

PROV has a diagrammatic representation that encapsulates relationships and tagging between the nodes which add reasoning and meaning to the raw data. PROV is built on logic and has a representation that is machine process-able.

The usefulness of mining data to answer questions and draw conclusions is determined by the ability of finding hidden patterns and anomalies in the data. Hence, a query platform will enable the mining of data and patterns for reasoning. It would be observed later that without the proposed framework it would be difficult to audit and find the causes of overload in data centres.

The challenge that faces this research is the validity of the assumption that provenance adds richness in data for log data analysis to help find the causes for machines overload due to over-commitment, explained in Chapter 2. With this in mind, a provenance-driven diagnostic framework [138] has been developed using Google Cloud 29-Day usage dataset for exploration. Its goal is to identify the causes for task evictions. As explained in Chapter 2, Task Eviction (TE) is one of six overload mitigating strategies. The framework extends the W3C PROV model [9] into PROV-Task Eviction (PROV-TE) which underpins a number of diagnostic algorithms for identifying evicted tasks due to specific causes.

4.3 Google Cloud 29-day Usage Dataset

Real production and usage datasets (also named workloads, log data and trace logs) play a pivotal role in conducting research. These real datasets support a wide variety of research domains and provide creditability and assurance for building models and designing simulation as they reflect realistic scenarios [139].

The biggest publicly available Cloud dataset, Google's Cloud 29-day usage dataset, has been utilized to extend the PROV model into PROV-TE because it applies Task Eviction mitigating strategy [7], [140]. As illustrated in Table 4.1, the dataset consists of a 29-day trace of its applications with over 25 million tasks grouped into over 650 thousand jobs running across over 12 thousand

heterogeneous machines from a Google data centre. The trace starts at 19:00 EDT on Sunday May 1, 2011. The documentation of the dataset [7] shows that Google applies over-commitment mechanism to utilize their resources. According to [5], [37], each task is scheduled in a Linux container which is a lightweight virtual system mechanism equivalent to a VM, as explained in section 2.2.5. The documentation also states that Google applies a Task Eviction mitigating strategy in order to mitigate the overload of usage (trade-off of over-commitment). The documentation states five causes for evicting tasks which are:

Cause 1. Take Over by Higher Priority Tasks

Cause 2. Increase In Resource Requests

Cause 3. Demand exceeds Physical Capacities

Cause 4. Missing Machines

Cause 5. Decrease in Machines Capacities

Table 4.1 Dataset Profile

Dataset period	29-day
Number of unique users	930
Number of unique tasks	25,405,064
Number of unique jobs	671,679
Number of unique physical machines	12,583

This dataset is made up of six database tables, which add up to the size of about 200 GB, namely Machine Event (ME), Machine Attribute (MA), Job Event (JE), Task Constraint (TC), Task Event (TEv) and Task Usage (TU). Each table has a primary key as an index and includes a timestamp. Also, each table is packaged

Table 4.2 Overview of the Dataset [160]

Table	ME	MA	JE	TEv	TC	TU
Number of csv file(s)	1	1	500	500	500	500
Data entries	37,780	10,748,566	5,012,242	144,648,288	28,485,619	1,232,792,102
Ave entries/file	37,780	10,748,566	4,024.5	289,296.6	56,971.2	2,465,584.2
Number of data parameters	6	5	8	13	6	20
Compressed size	339 KB	136 MB	83 MB	1.5 GB	147 MB	36.6 GB
Uncompressed size	2.77 MB	1.12 GB	315 MB	15.4 GB	2.82 GB	158 GB

in one or more CSV files and has a specific set of attributes (parameters). Summaries of the overall statistics of the tables as well as the list of all parameters are illustrated in tables 4.2 and 4.3, respectively.

ME and MA tables describe the machines characteristics, i.e. CPU capacity, and the status of the machine: Added, Updated, or Removed. JE and TEv tables are related to jobs and their tasks. They describe specific information such as status, i.e. submitted or killed, the priority, the level of sensitivity of the tasks and jobs. describes jobs and their lifecycle. There are 9 event types (status) of every job and task: Submit, Schedule, Evict, Fail, Finish, Kill, Lost, Update Pending, Update Running. They also show the amount of resources requested which can be used together with the machines capacity to identify the level of wasted resources. For example, 10 tasks were scheduled into machine A. The sum of the requested RAM is 100 units and the capacity of the machine's RAM is 100 units. When calculating the available RAM of the machine at run time, it is found that 30 units are available which were already allocated to the 10 tasks but not used, hence wasted. Moving on, TC table describes the task placement constraints that restrict the machines onto which tasks can schedule. Finally, TU

table is the biggest table and it describes the tasks' usage and resources consumption.

Table 4.3 Dataset Parameters

Dataset Tables	Parameters	Description of Table
Machine Events (ME)	1. timestamp 2. machine ID 3. event type 4. platform ID 5. capacity: CPU 6. capacity: memory	Each machine is described by one or more records in the machine event table. There are three types of machine events: ADD, REMOVE, and UPDATE.
Machine Attributes (MA)	1. timestamp 2. machine ID 3. attribute name: an opaque string 4. attribute value: either an opaque string or an integer 5. attribute deleted: a Boolean indicating whether the attribute was deleted	Machine attributes are key-value pairs representing machine properties, such as kernel version, clock speed, and presence of an external IP address. Tasks can specify constraints on machine attributes.
Job Events (JE)	1. timestamp 2. missing info 3. job ID 4. event type 5. user name 6. scheduling class 7. job name 8. logical job name	The job event table describes jobs and their lifecycle. There are 9 event types that describe the status of every job and task: SUBMIT, SCHEDULE, EVICT, FAIL, FINISH, KILL, LOST, UPDATE PENDING, UPDATE RUNNING.
Task Events (TEv)	1. timestamp 2. missing info 3. job ID 4. task index - within the job 5. machine ID 6. event type 7. user name 8. scheduling class 9. priority 10. resource request for CPU cores 11. resource request for RAM 12. resource request for local disk space 13. different machine constraint	The description of Job Events applies here. Tasks have the same lifecycle and represented in the dataset with 9 different status types recorded in the event type parameter. Event types SCHEDULE and EVICT are the focus of this thesis.

Task Constraints (TC)	1. timestamp 2. job ID 3. task index 4. attribute name -- corresponds to machine attribute table 5. attribute value -- either an opaque string or an integer or the empty string 6. comparison operator	This dataset describes the task placement constraints that restrict the machines onto which tasks can be scheduled.
Task Usage (TU)	1. Start time 2. End time 3. job ID 4. task index 5. machine ID 6. CPU usage 7. memory usage 8. assigned memory 9. unmapped page cache memory usage 10. page cache memory usage 11. maximum memory usage 12. disk I/O time - mean 13. local disk space used - mean 14. CPU rate - max 15. disk IO time - max 16. cycles per instruction (CPI) 17. memory accesses per instruction (MAI) 18. sampling rate 19. aggregation type 20. CPU sampling rate	Describes the tasks' usage and resources consumptions on a 5-minute logging interval.

In terms of distinguishing physical machine and virtual machines, the dataset states only one machine ID in MA table but does not state the type of the machine. In this research, during the initial development of the algorithms, there was no separation between the VM and PM due to the lack of information. However, during the evaluation of the algorithms using simulated environment, the algorithms were enhanced to consider both VMs and PMs, see sections 6.6.1 and 6.7.1.

In terms of the relationships between the parameters, they already normalized by the vendor and structured [7], hence being categorized in six tables. Also, each table comes with primary key or a compound key.

4.4 The Generic Framework for Provenance-Driven Diagnostic Model

The diagnostic framework can be divided into three phases, shown in Figure 4.1.

The goal of this framework is to map the raw data to a PROV model (phase 1),

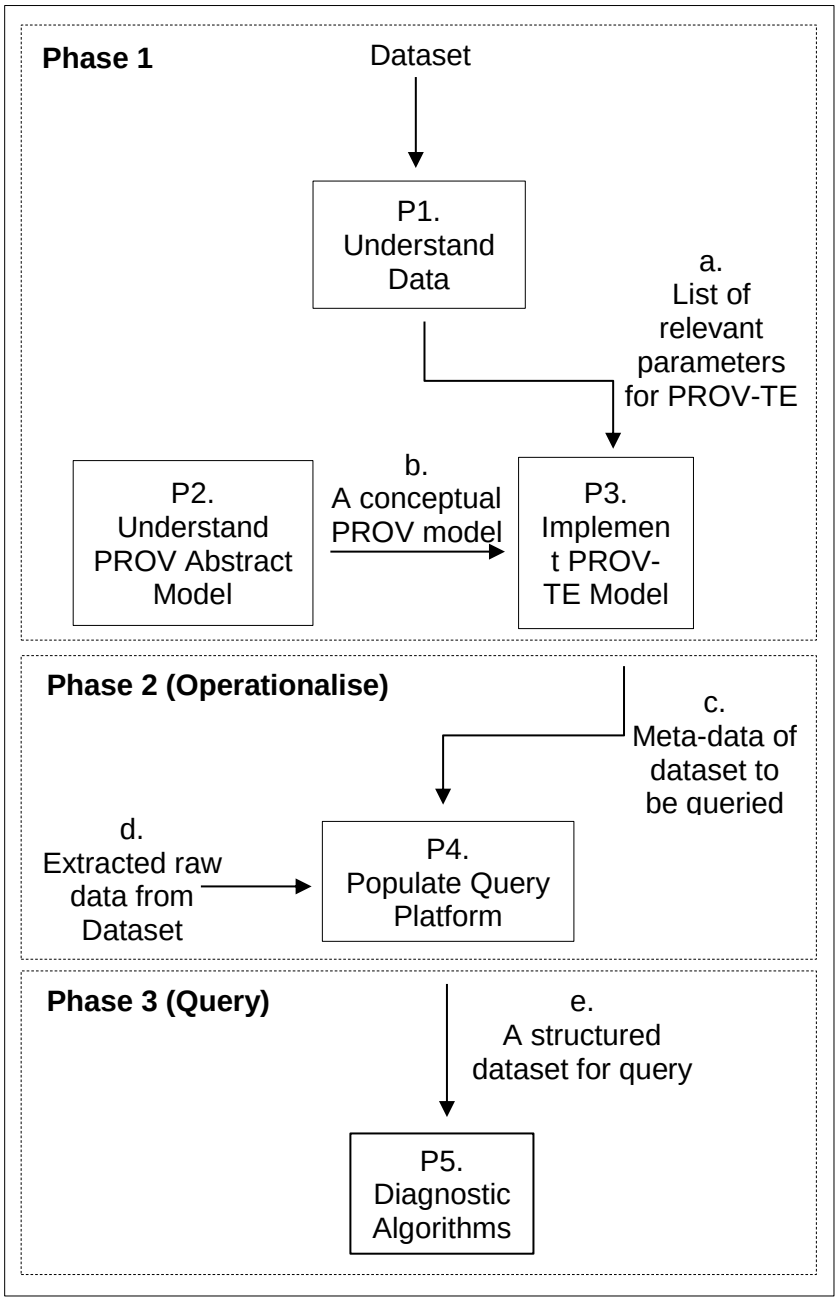

Figure 4.1 Steps for the Provenance-Driven Diagnostic Framework

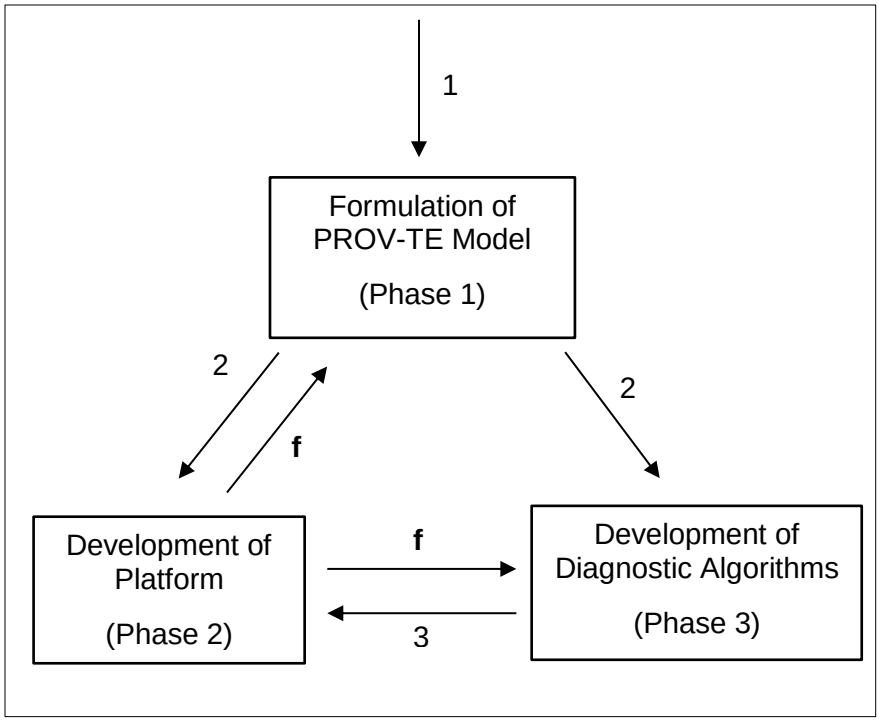

Figure 4.2 Iterations for Framework Improvement

operationalize it (phase 2) and to have a platform ready for querying and finally to develop algorithms for the queries (phase 3). This generic process can be used to develop specific diagnostic frameworks for different overload mitigating strategies depending on the available datasets.

The three phases will be described in detail in Sections 4.6 – 4.8.

4.5 Iterative Approach in Framework Development

Each Cloud data centre has unique configurations which makes usage datasets to be in different structure with different parameters. Similar to software development, refinement could be made based on feedback. Figure 4.2 shows the iterations that took place for this research. (1) Raw dataset helps formulate the model. (2) The model is then used to construct both the testing platform and the diagnostic algorithms. (3) The algorithms are then applied on the constructed platform for testing. (f) is the feedback from testing which informs both the model and the algorithms for further enhancements. The model, Figure 4.3, has gone

through two iterations informed by the feedback. The continuous feedback while testing helped in constructing the algorithms. A number of SQL queries were merged into one algorithm. Next, the framework is discussed in detail.

4.6 Phase 1: PROV-TE Formulation

The input of this phase is the Google 29-day Cloud Usage dataset, presented in Section 4.3. Data in Google's dataset is structured and huge in volume. Not all data are relevant, hence a subset will be selected to enable further analysis and minimize processing time.

The first process of this phase (P1) starts with understanding the meaning of the attributes (parameters) in the dataset tables by consulting the publications of Google on the datasets.

Process 2 (P2) relates to understanding the abstract model of PROV, presented in Chapter 2. The PROV model consists of edges and nodes. Nodes can be one of the following: Entity- a digital, conceptual or physical thing of which we need to keep the provenance; Activity- a process that occurs over a duration of time that act upon entities; and Agent- something/someone to which entities and activities are attributed or associated. Edges represent the dependencies between these nodes; for instance, prov:Used, prov:WasGeneratedBy, prov:WasDerivedFrom, prov:WasAssociatedWith, prov:WasAttributedTo, prov:WasActedOnBehalf, and prov:WasInformedBy. The output of these two processes (P1 and P2) are (a) an understanding of the meaning of the parameters in the dataset which led to specifying the needed parameters for PROV-TE and (b) an understanding of the skeleton of PROV model.

Table 4.4 Selected Parameters for PROV-TE

PROV Nodes	Parameters	
Agent	• User, • Scheduler	
Activity	• Submit task, • Group Tasks into jobs, • Schedule Task/Job, • Add Machine, • Update Machine, • Update Pending/Running, • Evict Task/Job	
Entity (Selected Parameters)	• JE_timestamp, • JE_jobID, • JE_eventtype, • JE_username • TEv_timestamp • TEv_taskindex, • TEv_eventtype, • TEv_priority, • TEv_username, • TEv_schedulingclass, • TEv_resource_request_CPU, • TEv_resource_request_RAM, • TEv_differentmachine,	• ME_timestamp, • ME_machineID, • ME_eventtype, • ME_capacityCPU • ME_capacityRAM • MA_timestamp, • MA_attributename • MA_attributevalue • MA_attributedeleted
Unused Parameters	• TC_timestamp • TC_jobID • TC_taskindex • TC_comparison_operator • TC_attribute_name • TC_attribute_value • TU_starttime • TU_endtime • TU_jobID • TU_taskindex • TU_machineID • TU_CPU_usage • TU_memory_usage • TU_assigned_memory_usage • TU_unmapped_page_cache_memory_usage • TU_page_cache_memory_usage • TU_maximum_memory_usage • TU_disk_I/O_time_mean	• TU_local_disk_space_used_mean • TU_CPU-rate_max • TU_disk_IO_time _max • TU_cycles_per_instruction • TU_memory_accesses_per_instruction • TU_sampling_rate • TU_ggregation_type • TU_CPU_sampling_rate

Process 3 (P3) is the implementation of PROV-TE (PROV Task-Eviction) according to the list of selected parameters. Having understood the parameters from the trace's documentation, only the needed ones that capture the data

relevant to task eviction are selected, shown in Table 4.4[1]. P3 takes (a) and (b) as input and the output of this process is PROV-TE model which provides (c) the meta-data of the dataset to be queried.

4.6.1 The PROV-TE Model

Figure 4.3 shows a diagrammatic representation of PROV-TE, the output of this phase. It is the second version of PROV-TE. The first version of PROV-TE is shown in [138], details are presented in Appendix A.

PROV-TE model was constructed by (i) putting together the workflow processes of scheduling a task and the five causes explained in sections 4.6.2 and 4.6.3, (ii) including the relevant parameters (entities) explained in Table 4.4, (iii) mapping the dependencies between the Activity, Agent and Entity of every workflow based on the generic PROV model, and (iv) following the normalization of the dataset tables.

[1] The agents and the activities have been captured from the dataset's documentation but not recorded in the dataset itself.

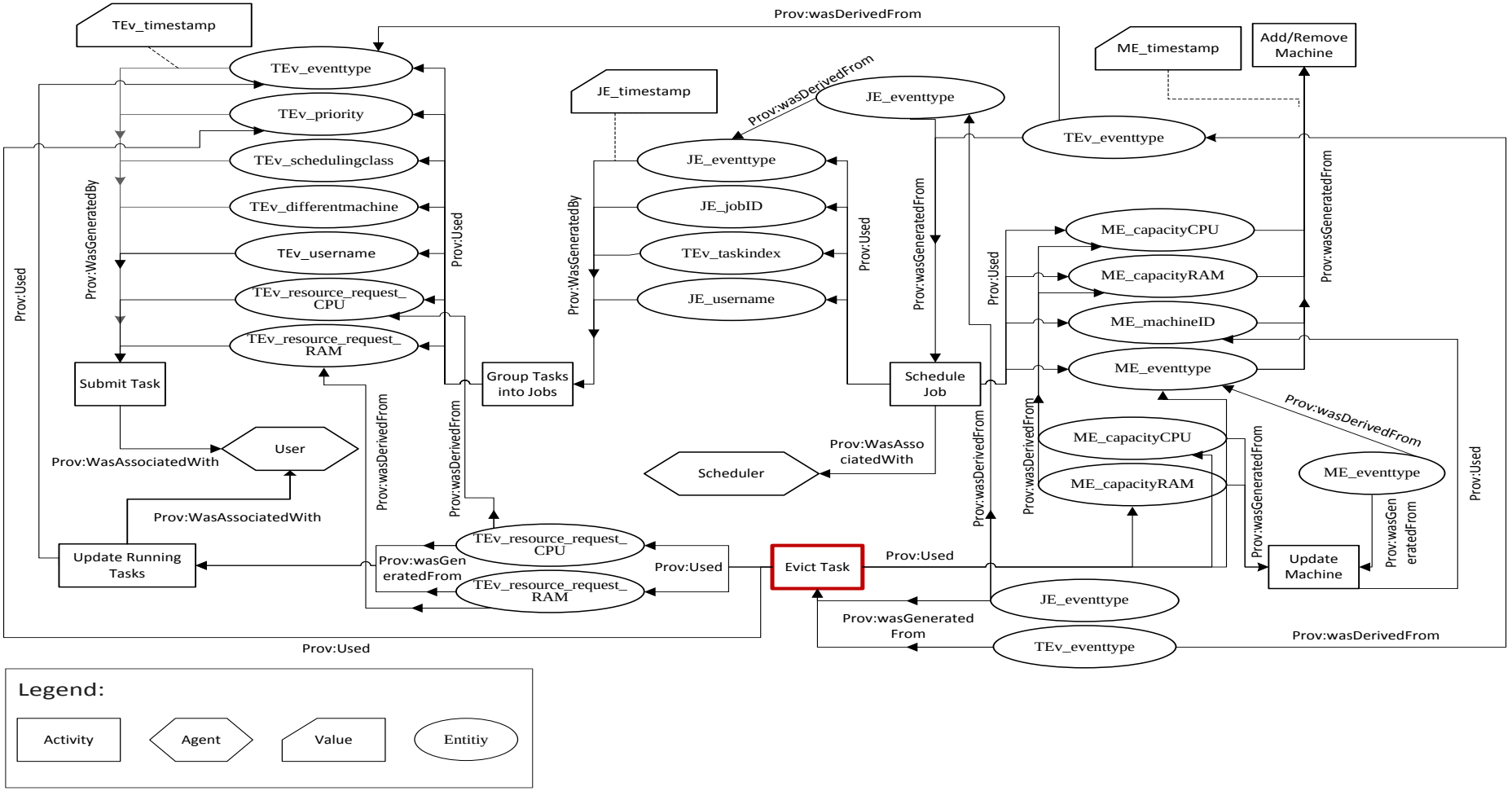

Figure 4.3 PROV-TE, a PROV Model for Task Eviction Mitigating Strategy

4.6.2 Workflow for a Scheduled Task

Following is an illustration of how the PROV-TE model can be used to trace the workflow of a task from user submission *Agent: User* to being hosted by *Agent: Scheduler*, refer to Figure 4.3. Normally, an *Agent: User* submits a task and specifies its scheduling priority *Entity: TEv_priority*. After a task is submitted *Activity: Submit Task*, a number of Entities are generated, i.e. *TEv event type, TEv priority, TEv resource CPU/RAM*, and all have a time stamp. Those entities are used by the *Activity: Group Tasks into Job*. Then a number of Entities are generated according to the grouping activity, i.e. *JE jobID, JE event type, JE job name, and TEv task index and JE time stamp* is recorded. The *Activity: Schedule Job* will use those entities and other entities related to the designated Machine, i.e. *ME_MachineID, ME_eventtype, ME_capacity CPU/RAM,* so that the task/job can be scheduled and hosted.

4.6.3 Task Eviction Workflows

This section illustrates how to trace task eviction workflows in PROV-TE model with respect to the five causes. Knowing task eviction workflows beforehand helps in the development of the diagnostic algorithms.

Activity: Evict Task constantly checks (*Prov: Used)* the following entities: *Entity: ME_capacityCPU, Entity: ME_capacityRAM, Entity: ME_eventtype, Entity: TEv_resource_request_CPU* and *Entity: TEv_resource_request_RAM*. It gets triggered to process the eviction of one or more lower priority tasks i.e. *Entity: TEv_priority* on the same machine *Entity: ME_machineID* if one or more of the following scenarios takes place:

Cause 1. a new task with a higher priority *Entity: TEv_Priority* is submitted to a machine *Entity: ME_machineID* that lacks resources or encounters a temporary loss of resources Entity: *ME_capacityCPU* and *Entity: ME_capacityRAM*, see Figure 5.5 in section 5.4.1.

Cause 2. *Agent: User* updates the request of resources *Entity: TEv_resource_request_CPU* and *Entity: TEv_resource_request_RAM* for the running tasks Activity: Update Running Tasks but there are no enough resources *Entity: ME_capacityCPU* and *Entity: ME_capacityRAM* to approve the new request, see Figure 5.6 in section 5.4.2.

Cause 3. The approved capacities requests *Entity: TEv_resource_request_CPU* and *Entity: TEv_resource_request_RAM* exceeds the machine's capacity *Entity: ME_capacityCPU* and *Entity: ME_capacityRAM* at any point of time, see Figure 5.7 in section 5.4.3.

Cause 4. A machine is Removed in *Entity: ME_eventtype* and the already scheduled tasks *Entity: TEv_taskindex* and *Entity: jobID* can no longer be accommodated, see Figure 5.10 in section 5.4.4.

Cause 5. A machine status is Updated in *Entity: ME_eventtype* and the new physical or virtual capacities *Entity: ME_capacityCPU* and *Entity: ME_capacityRAM* are less than the approved resources requests of the already scheduled tasks *Entity: TEv_resource_request_CPU* and *Entity: TEv_resource_request_RAM, see Figure 5.11 in section 5.4.5*.

4.7 Phase 2: Preparation of Platform for Queries

The input of this phase is the meta-data provided by the extended PROV model, PROV-TE, refer to (c) in Figure 4.1. The meta-data is used to build the data model storage which supports the querying mechanism. For this research, a relational database with structured and cleaned dataset, the output of this phase, has been developed for this framework because a lightweight provenance model is applied. Other types of databases could also be used such as graph database. For a heavyweight provenance model, semantics platform could be developed which supports querying mechanisms such as SPARQL.

While investigating the causes, one by one during first iteration, it was discovered that some parameters were missing in order to conduct the investigations. They provided the feedback to phases 1 and 3 for improvements (as represented by (f) in Figure 4.2).

4.8 Phase 3: Diagnostic Algorithms Formulation based on PROV-TE

This phase takes the processed dataset as input ((e) in Figure 3,.1) and use the constructed querying platform in order to develop and test the diagnostic algorithms, (process P5 in Figure 4.1).

In this thesis, it is assumed that a task can be evicted due to one or more causes, discussed in section 5.5. This suspension will be tested by the application of the algorithms in chapter 5. The 5 causes identified from the literature were adopted for investigation. 10 diagnostic algorithms have been developed based on PROV-TE to facilitate the querying process. Causes 1 – 5 are investigated

separately and diagnosed by a set of algorithms that are implemented using SQLite. The following is a discussion on the developed diagnostic algorithms for the 5 causes presented in section 4.3.

4.8.1 Arrival of Higher Priority Tasks

Table 4.5 Algorithm 1a: Cause 1 Priority Identifier

Finding the priority of evicted tasks and isolating the tasks in a separate table (PriorityofEvictedTasks).
1. FOR each task in TaskEvents table (TEv), until end of period **2. IF** status = Killed **3. STORE** distinct TEv_taskindex, TEv_priority and TEv_timestamp in *PriorityofEvictedTasks table (PET).* **4. END IF** **5. END FOR**

Table 4.6 Algorithm 1b: Cause 1 Eviction Identifier.

Identifying the number of evicted tasks from PriorityofEvictedTasks table (PET) within one-step interval from higher priority tasks being scheduled in the same machine.
1. FOR each task in PET, until end of period **2. FOR** each task in TaskEvent (TEv) table, until end of period **3. IF** ((TEv.timestamp < PET.timestamp <= (TEv.timestamp+ next time interval) **AND** (PET.priority < TEv_priority) **AND** (PET_machineID = TEv_machineID)) **4. STORE** distinct PET.Task in *Cause1EvictedTasks table* **5. END IF** **6. END FOR** **7. END FOR**

One of the causes of task eviction is due to higher priority tasks taking over the space of the lower priority ones upon scheduling. This trigger is due to the VMs' limited resources. With help of PROV-TE, two algorithms have been used to investigate this scenario. First, all evicted tasks in the dataset are captured and their priorities are ordered and stored (Algorithm 1a). The aim is to precisely identify the tasks the have been evicted only by Higher Priority Tasks being scheduled in the same Host (VM) and within one interval of higher priority task arrival timestamp (Algorithm 1b). PROV-TE has helped in constructing these algorithms following the workflows mentioned in section 4.6.

4.8.2 Increase in Resource Requests

Table 4.7 Algorithm 2a: Cause 2 Request Comparer**.**

Comparing the resources' request of both CPU and MEM at the task's scheduling time against the new resources' request while running, then identify the tasks with the increased update of resources' request and isolate them in Updated table (UT).

1. FOR each task in TaskEvent (TEv) table, until end of period
2. **IF** (Status = scheduled (S)
AND updated_while_running (U) = true
AND ((TEv_resource_request_CPU of U >
TEv_resource_request_CPU of S)
OR (TEv_resource_request_Mem of U >
TEv_resource_request_Mem of S)))
3. **STORE** Task_timestamp, Task ID, machineID in *Updated Table (UT)*
4. **END IF**
5. END FOR

Another cause of task eviction is when users ask for more resources than they have initially requested while their tasks are running. Each task is scheduled in

Table 4.8 Algorithm 2b: Cause 2 Eviction Identifier.

Looking within the lowest granularity interval of the dataset, one interval, from the time of the task resources' request update in Updated Table (UT) to identify the tasks that have been evicted due to the increase in the update.

```
1. FOR each task in UT table, until end of period
2.     FOR each task in TaskEvent table (TEv) with an increase to their
       resources' request, until end of period
3.         IF ((TEv.Status = evict)
           AND (Task_timestamp (UT) < Task_timestamp (TEv) <=
           (Task_timestamp (UT) + next time interval))
           AND Task priority (UT) > Task priority (TEv))
4.             THEN display TEv.Task ID, Task_timestamp
5.         END IF
6.     END FOR
7. END FOR
```

a specific VM with specific virtual resources (assigned resources according to their request). In the case of over-commitment , when users request more resources, the scheduler neither can allocate more resources nor find an available virtual machine. A physical machine with fixed resource capacity would no longer be capable of continuing to host those tasks because the sum of the tasks' virtual resources' usage could get higher than the actual machine's capacity. So, lower priority tasks get evicted to avoid an overload in the machine. The aim of algorithms 2a-b is to find the evicted tasks because of such scenario.

4.8.3 Demand Exceeds Physical Capacities

Resources over-commitment causes overload [2], [4]. Cloud providers set a usage threshold level where once it has been reached, an overload mitigating strategy, i.e. Task Eviction, is then triggered [2]. However, in Google's case [7] the threshold level is not documented nor stated in the dataset. Thus, algorithms 3a-b have been developed to identify the total capacities of the physical

Table 4.9 Algorithm 3a: Cause 3 Capacities Calculator

Identifying the total capacities of the machines and the total requested resources (CPU and MEM) per day for the period of the trace (29 days).

1. **FOR** all available physical machines in machine_events table and all requested resources of tasks in task_events table, until end of period
2. **Sum** the total capacity of ME_capacityMEM and ME_capacityCPU
3. **Sum** the total capacity of TEv_requestedCPU, TEv_requestedMEM
4. **Group By** Day
5. **STORE** Day Number**,** TEv_requestedCPU, TEv_requestedMEM, ME_capacityMEM and ME_capacityCPU in *Overload Table (OT)*
6. **END FOR**

machines and the total assigned resources in a daily basis. Then, the day that has higher resources request than the actual physical capacity is assumed to result in an overload. Tasks evicted on those days are presumed to be the result of the occurred overload instances.

Table 4.10 Algorithm 3b: Cause 3 Eviction Identifier

Identify the days where the requested resources (MEM and CPU) are higher the available physical resources from the Overload Table (OT).

1. **FOR** every day in *Overload Table (OT)*, until end of period
2. **IF** ((ME_capacityMEM < TEv_requestedMEM) **OR** (ME_capacityCPU < TEv_requestedCPU))
3. **Display** tasks with status = evicted
4. **END IF**
5. **END FOR**

4.8.4 Missing Machines

The fourth investigation looks at the dataset from the physical machine point of view. The attribute ME_eventtype of PROV-TE tells whether the machine is Add, Update, or Remove. Removal of physical machines can usually be caused due

to maintenance or failure. Thus, tasks that have been scheduled to run on those machined get evicted.

Table 4.11 Algorithm 4a: Cause 4 Removal Identifier

Finding machines with a Remove event type and storing their IDs in a separate table.
1. FOR each machine in machine_events table, until end of period **2.** **IF** ME_eventtype = remove **3.** **STORE** ME_machineID, ME_timestamp in *Removed Machine IDs table (RMI)* **4.** **END IF** **5. END FOR**

Table 4.12 Algorithm 4b: Cause 4 Eviction Identifier

For every removed machine in Removed Machine IDs table (RMI), identify the tasks that have been evicted within on-interval of the removal.
1. FOR each task scheduled in RMI, until end of period **2.** **IF** (TEv_eventtype = evict) AND **3.** (RMI.ME_timestamp < TEv_timestamp <= (RMI.ME_timestamp + next time interval)) **4.** **Display** distinct JE_JobID, TEv_taskindex **5.** **END I**F **6. END FOR**

4.8.5 Decrease in Machines Capacities

For various reasons, machines' capacities get reduced. That reduction can cause tasks to be evicted. Algorithm 5a identifies the physical machines which have encountered an update to their resources in their lifetime. The event type attribute in Machine Events table (ME_eventtype) is used. Every machine in the log-data has three event types, Add, Remove, or Update. The ME_eventtype 'Update' allows us to exactly identify the updated machines and work out if the

resources has been reduced or not by comparing with its capacity at previous state ‘Add’. Then, Algorithm 5b looks at the impact of such reduction by identifying the evicted tasks within one interval from the update timestamp.

Table 4.13 Algorithm 5a: Cause 5 Removal Identifier

Comparing the capacity of the machines at add event and at update event, then storing IDs of machines with decreased capacities in *Machines Decreased Capacity table (MDC).*
1. FOR each machine in machine_events table, until end of period **2.** **IF** ME_eventtype = add AND update • Compare capacity at add with capacity after update for ME_capacityCPU , ME_capacityRAM **3.** **IF** ((ME_capacityCPU(at add) > ME_capacityCPU (after update)) OR (ME_capacityRAM(at add) > ME_capacityRAM (after update))) **4.** **STORE** ME_timestamp, ME_machineID, ME_capacityCPU, ME_capacityRAM in *MDC table*. **5.** **END IF** **6.** **END IF** **7. END FOR**

Table 4.14 Algorithm 5b: Cause 5 Eviction Identifier

Looking within the lowest granularity interval of the dataset from the time of the machine update in order to identify the tasks that have been evicted due to the decrease in the machine capacity.
1. FOR each machine in MDC table **2.** **FOR** each task in task_events table scheduled in machines with decreased update, until end of log data **3.** **IF** (TEv_eventtype = evict) AND **4.** (ME_timestamp < TEv_timestamp <= (ME_timestamp + next time interval) **5.** **DISPLAY** TEv_timestamp, JE_JobID, TEv_taskindex, ME_timestamp, ME_machineID **6.** **END IF** **7.** **END FOR** **8. END FOR**

4.9 Instantiation of the Framework – The Auditor

As explained in Chapter 2, there are three delivery models for Cloud services; SaaS, PaaS, and IaaS. The framework's application is in the IaaS layer. A proof-of-concept system, Auditor, was developed and its positioning is shown in Figure 4.4. It shows how the framework can be applied to the reality of research and engineering. The Auditor consists of three components: Mapper, Database, and Query Handler. The Mapper takes the raw data from the dataset gathered by the Infrastructure Monitor component as input and maps it to the PROV-TE model structure which then is stored in the database. The Query Handler is the implementation of the diagnostic algorithms discussed earlier. It gets the structured dataset from the database as input, runs the algorithms using SQLite, and then informs the Virtual Infrastructure Manager (VIM) with the causes of TE. The VIM or the Cloud provider could make use of the Auditor to make decisions.

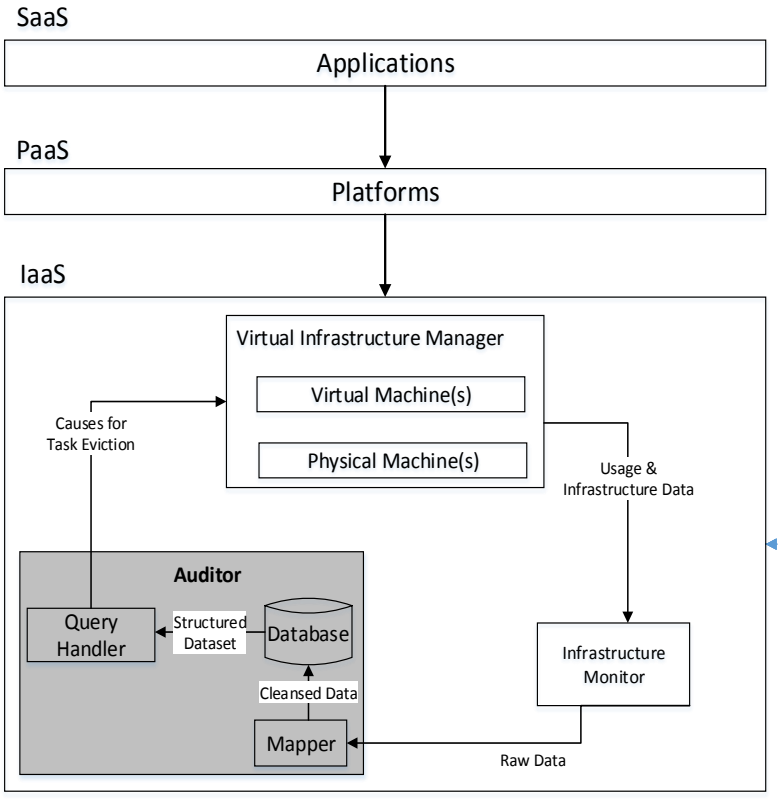

Figure 4.4 System Model

The potential use of the proposed framework is that following the process of framework development, the other five mitigating strategies could be modelled based on PROV and the relevant diagnostic algorithms could be developed. As a result, each mitigating strategy could have its own Auditor; e.g. Auditor for Live Migration causes, Auditor for VM Quiescing causes and so on.

In addition, users could make use of the auditor as a third party framework to audit the Cloud services they use and the providers management of the data centre. Because users lack the accessibility to the providers monitoring tools, the auditor could potentially be facilitated for this purpose. For example, users might be able to understand and know the exact reason behind why their tasks are not finishing within the expected timeframe.

4.10 Summary

This Chapter has presented the novel provenance-driven diagnostic framework and the methodology followed to construct it. The underpinning philosophy and assumptions of the framework have been explained. The real Cloud dataset used for learning has been analysed in detail and presented.

The three phases of the formulation methodology the provenance-driven diagnostic framework for task eviction have been explained. Working scenarios of tracing of task eviction workflow demonstrating PROV-TE model have been illustrated. Further, the diagnostic algorithms of all five causes of task eviction which are used identify the evicted tasks and the relevant causes have been presented and discussed. In the next Chapter, the application of the framework on a real Cloud dataset will be presented and discussed in detail.

Chapter 5

Application of the Diagnostic Algorithms

5.1 Introduction

This chapter presents the application of the developed diagnostic algorithms. It investigates how PROV-TE could diagnose and audit Task Eviction causes from a given Cloud usage dataset. An exploratory experiment was set up to investigate the causal relationships between the causes and Task Evictions using the proposed diagnostic algorithms in the framework. It starts with presenting the context of the experiment. Then, it discusses the aim and hypothesis of the experiment. It then illustrates the application of the proposed diagnostic algorithms over the given Cloud usage dataset. It describes how PROV-TE contributes to every investigation. It ends with a summary of the overall findings of the experiment.

5.2 Context for the Experiment

As explained in section 4.3, using real datasets in research adds assurance and credibility in the results and findings [139]. Where there are assumptions, the use of real datasets reflect the realistic scenarios in verifying and proving such assumptions. Google's 29-day Cloud Usage dataset has been used for this exploratory experiment.

SQLite is a relational database management system and was used as the query platform. This section discusses how the datasets were restructured into relational tables. The Google 29-day trace dataset in six folders, one for each

dataset table. Each folder contains up to 500 csv files. Using Linux command *cat *.csv >> output.csv*, all files in each folder have been merged into one csv file. After merging the relevant files together, we end up with six csv files, each represent a dataset. Four csv files have been imported into four separate database tables, namely Machine Events, Machine Attributes, Job Events and Task Events. The diagnostic algorithms, explained in Section 4.8 have been translated into SQLite queries. Figures 5.1 – 5.4 are snapshots of the first 10 rows of the used tables; machine_events table, machines_attributes table, job_events table, and task_events table.

```
sqlite> select * from machine_events limit 10;
timestamp   machineID   eventtype   platformID                                    capacityCPU  capacityMEM
----------  ----------  ----------  --------------------------------------------  -----------  -----------
0           5           0           HofLGzk1Or/8Ildj2+Lqv0UGGvY82NLoni8+J/Yy0RU=  0.5          0.2493
0           6           0           HofLGzk1Or/8Ildj2+Lqv0UGGvY82NLoni8+J/Yy0RU=  0.5          0.2493
0           7           0           HofLGzk1Or/8Ildj2+Lqv0UGGvY82NLoni8+J/Yy0RU=  0.5          0.2493
0           10          0           HofLGzk1Or/8Ildj2+Lqv0UGGvY82NLoni8+J/Yy0RU=  0.5          0.2493
0           13          0           HofLGzk1Or/8Ildj2+Lqv0UGGvY82NLoni8+J/Yy0RU=  0.5          0.2493
0           14          0           HofLGzk1Or/8Ildj2+Lqv0UGGvY82NLoni8+J/Yy0RU=  0.5          0.2493
0           19          0           HofLGzk1Or/8Ildj2+Lqv0UGGvY82NLoni8+J/Yy0RU=  0.5          0.2493
0           21          0           HofLGzk1Or/8Ildj2+Lqv0UGGvY82NLoni8+J/Yy0RU=  0.5          0.2493
0           23          0           HofLGzk1Or/8Ildj2+Lqv0UGGvY82NLoni8+J/Yy0RU=  0.5          0.2493
0           25          0           HofLGzk1Or/8Ildj2+Lqv0UGGvY82NLoni8+J/Yy0RU=  0.5          0.2493
sqlite>
```

Figure 5.1 A snapshot of Machine Events table

```
sqlite> select * from machine_attributes limit 10;
timestamp   machineID   attributename                                 attributevalue  attributedeleted
----------  ----------  --------------------------------------------  --------------  ----------------
0           5           +8aLrMrUBQLYLD4Ndxi/ZmwwKGhtFGskL8ZSdChYMdo=  1               0
0           5           4chZqkETND7f4XrzdWLe8c1eLHKuYgjnUMSU+83EYFA=  1               0
0           5           5dEuieuWMFy+CNMBBf/uXNX5nP4Kgzeu0O6UizRNK3w=  2               0
0           5           81Cd14q4ebyI/3TPkkIwGYlB/mvKhBUk3e/E8egb4s0=  2               0
0           5           9eCGRtl6XN5GQoOYGEjKtupBbtUoOaOPYRFw+pzH7IU=  2               0
0           5           AbDt9LdIlA7DylAxXIG9AiJ2J18Cuhpmk5fouqfP1LQ=  1               0
0           5           ByISK6v8seuIxB23FGnamad97zwwKG2auw7Zs6h9Boc=  4               0
0           5           CPN9sCUKIBSqzdaCv2CNTsbDRt/oA6rDAUecPSbyeCY=  tHf5iHqQjCIQdZ  0
0           5           Drtjw4xPqRquKalsI2q5OaN14xXUha47Qs7m83yx0fQ=  1               0
0           5           Eqr5/diyFWeYBfm508mvBU9y9L8uPjCkYuL1DtgDaSU=  1               0
sqlite> _
```

Figure 5.2 A snapshot of Machine Attributes table

```
sqlite> select * from job_events limit 10;
timestamp   missinginfo  jobID       eventtype   username                                      schedulingclass  jobname                                       logic
aljobname
----------  -----------  ----------  ----------  --------------------------------------------  ---------------  --------------------------------------------  -----
--------------------------------------
0                        3418309     0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                IHgtoxEBuUTHNbUeUs4hzptMY4n8rZKLbZg+Jh5fNG4=  wAmgn
2H74cdoMuSFwJF3NaUEaudUBTZ0/HaNZBwIpEQ=
0                        3418314     0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                L52XDyhi9x9ChmUBZ1qavOFmnzPeUsvQ2QyGmBZcU4s=  ShNje
aoUeqGU2i9WMKEX9HTeuc9K2Fdfovibt7Mp6qI=
0                        3418319     0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                vq0IN3BWEbkDjYgYvkrUyH6OWoUoDwFFf3j/syEZzLA=  1A2GM
17AzHRcKJcJet/oIF7FOORyPcAOcUSpR9Fqou8=
0                        3418324     0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                X+Vce15Yu3BCKb7Ttc6hvINAzdfG3NtYEDNNsPdMGKo=  seczU
o7MBfi/kH3+eH/40Hxs012YKFtXFnamcvMjSQI=
0                        3418329     0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                EeK3DUWYi1P0vgBTp7wZdUos8UKj/+/FqudTLohMQ9M=  OEeQs
aVr4kdGHFwQ21iQlDZ18529HEMmu6B6/3K1cBA=
0                        3418334     0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                noCrQkR+8CU32ibuNmNvobHFFGuXRZ2aM/ZvcMHaOg4=  ys6UJ
Lz4Tz6SJxZJ7kwAZgxPgWnQAQiGIRIMcjOfU7I=
0                        3418339     0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                ORioZ5deSIAxIuzqmXo1Ivac+22YZbCucJEC0EqRbDc=  vnyDE
fuZHSuCdaC9/e75DUMRro/rqh96Eb4I6sy1Vlc=
0                        3418356     0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                QGf5Bi+7GCnOYjucoJ9xKNPFf9bWGxoEA+M1A1JPvbQ=  fGRnr
2XEPDr3kQsPccU/k1LELeeQonkj6hDpIP7ALkg=
0                        3418363     0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                HO9RaUGS/y7PpuPROMoq75iyAyG+zXqM9g6Tvy5fWH8=  sqjiC
yXrA7ELryW0hfrlWgso+S0gl2bzpI64QYJnrgc=
0                        3418368     0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                mIF6r5X6ITau4MPuBIE+QevbEjjACcfyUeRTAU791pg=  Uf/QC
CjtbryfYhlweVpii4xGkiY/9ml5CzU9Wz1DEY4=
sqlite> _
```

Figure 5.3 A snapshot of Job Events table

```
sqlite> select * from task_events limit 10;
timestamp   missinginfo  jobID       taskindex   machineID   eventtype   username                                      schedulingclass  priority    resourcerequest
CPU  resourcerequestMEM  resourcerequestDISK  differentmachine
----------  -----------  ----------  ----------  ----------  ----------  --------------------------------------------  ---------------  ----------  ---------------
---  ------------------  -------------------  ----------------
0           2            3418309     0           4155527081  0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                9

0           2            3418309     1           329150663   0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                9

0                        3418314     0           3938719206  0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                9           0.125
     0.07446             0.0004244            0
0                        3418314     1           351618647   0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                9           0.125
     0.07446             0.0004244            0
0           2            3418319     0           431052910   0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                9

0           2            3418319     1           257348783   0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                9

0           2            3418324     0           5655258253  0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                9

0           2            3418324     1           3550322224  0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                9

0           2            3418329     0           1303745     0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                9

0           2            3418329     1           3894543095  0           70s3v5qRyCO/1PCdI6fUXnrW8FU/w+5CKRSa72xgcIo=  3                9

sqlite>
```

Figure 5.4 A snapshot of Task Events table

5.3 Hypothesis and Aim

The hypothesis for this exploratory study is that PROV-TE, an extension of the provenance model PROV, provides adequate reasoning support for auditing Google dataset for the causes and timing of Task Eviction.

Google has stated the causes for Task Eviction (see section 4.3) but their extent of impact has neither been quantified nor mentioned. The aim of this experiment is to test the hypothesis that provenance, represented here by PROV-TE, adds value to the raw data by injecting the meaning and relationship between the data for log data analysis.

5.4 Applying the Diagnostic Algorithms

PROV-TE is used to guide the investigations into job/task behaviour leading to the stated causes for task eviction as suggested by Google. The investigations start after going through the three phases of the Provenance-Driven Diagnostic Framework as explained in sections 4.4 – 4.8.

Each one of the five causes is in a separate investigation. Each investigation is to see how PROV-TE help identify tasks that have been evicted by a specific cause. After the identification of the evicted tasks, questions arise, such as why that cause have happened? Also, where exactly that cause took place? Pinpointing the exact evicted tasks and tracing backwards the provenance of such instances with the help of PROV-TE could answer such questions.

The number of evicted tasks due to every cause is unknown at this stage. In terms of evictions, out of 25,242,731 tasks in the dataset, it has been found that

1,422,317 tasks were evicted. This has been found by counting the distinct evicted tasks in the dataset using the parameter TEv_eventtype with the value EVICT, regardless of the causes. The algorithms will look into the 1,422,317 evicted tasks particularly and try to identify the exact causes. According to the documentation of the dataset [7], the evictions can be because of a machine (i.e. Cause 4) or another task (i.e. Cause 1). While conducting the investigations, any found limitation will be recorded and discussed later in this Chapter. The expectation is that the total number of evicted tasks to be found by the application of the diagnostic algorithms will be equal to the total number found before conducting the investigations, 1,422,317.

5.4.1 Investigation 1 (Evicted Tasks caused by Take Over of Higher Priority Tasks)

This investigation looks at the priority of tasks. In the case when all physical resources are fully allocated and a new task is submitted with higher priority than those already scheduled, one or more lower priority tasks will get evicted. The selection of the exact task to be evicted is unknown. This is the process Google follows with regards to task eviction [7].

This investigation is to identify the tasks that are linked to Cause 1 Take Over by Higher Priority Tasks to be scheduled.

There are 12 priorities logged in the dataset, ranging from 0 to 11 as the bigger the number the more important the task is. In case of an overload, Google's only mechanism is to evict the lower priority tasks as higher priority tasks get preference for resources over lower priority ones. According to the dataset's

supporting document, there are three categories of task priorities. First, free priorities. This category has the least importance over the rest, Second, production priorities where tasks fall in this category have the highest priorities. Tasks that are latency sensitive are prevented by the scheduler from being evicted. Lastly, the third category is monitoring priorities, where higher priority jobs monitor lower priority ones. However, it is not possible from using both the documentation and the dataset to map the 11 priorities to the three categories. Table 5.1 shows the priority distribution of tasks in the dataset.

Table 5.1 Priority Distribution

Priority Number	Number of Tasks	Percentage of Total Tasks
0	6472128	25.48%
1	2453482	9.66%
2	1111810	4.38%
3	1027	0.004%
4	14197733	55.89%
5	104	0.0004%
6	639784	2.52%
7	400	0.002%
8	254680	1%
9	286269	1.13%
10	1403	0.01%
11	7538	0.03%

5.4.1.1 Contribution of PROV-TE

A number of attributes have been identified in order to undertake this investigation which are TEv_priority, TEv_eventtype and TEv_timestamp, shown

in Figure 5.5. TEv_priority specifies the priority of the task. TEv_eventtype specifies the status (e.g. SCHEDULE, EVICT, FAIL) of the tasks. It has been used to only count the scheduled and the evicted tasks while using the third

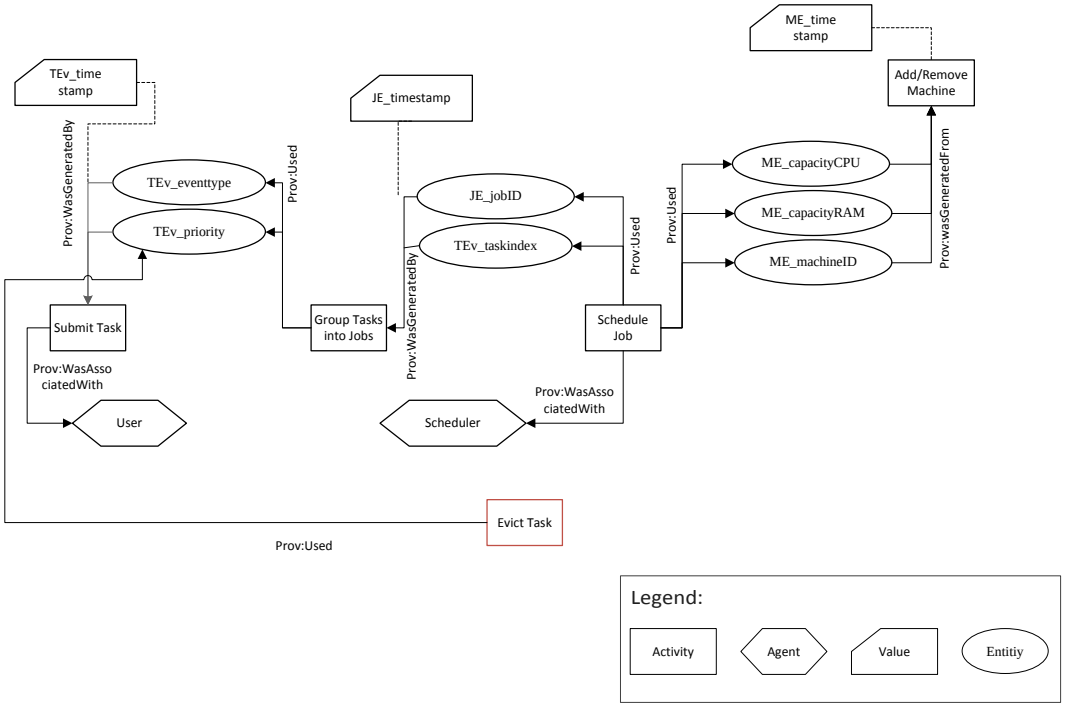

Figure 5.5 PROV-TE for Take Over by Tasks with Higher Priority

attribute, TEv_timestamp, to only look at the 5-minute interval because this is the logging interval of the dataset. So, any task that is evicted within one logging interval from a higher priority task being scheduled is considered in this investigation.

5.4.1.2 Results and Analysis

In this investigation, algorithms 1a and 1b, see tables 4.5 and 4.6 in Chapter 4, have been applied on the dataset which identify the evicted tasks due to the arrival of higher priority tasks into the host machine.

Out of a total of 25 million tasks in the 29-day dataset, 18,693,472 distinct tasks were scheduled with higher priority that those already running (Cause 1). Also, it has been found that a total of 1,421,054 distinct tasks were evicted in the same host machine.

Hence, there is a possible link between these 18,693,472 (76% of the total tasks) to the eviction of 1,421,054 lower priority tasks to be evicted due to lack of resources (Cause 0).

5.4.2 Investigation 2 (Evicted Tasks caused by Increase In Resource Requests)

When a task is submitted a number of parameters are set, such as priority, CPU and Memory. This determines where the tasks can be scheduled and how quickly. The requested capacities of a running task can be amended on the fly by the scheduler fulfilling the requests of users. When there is not enough resources and there are lower priority tasks, the task eviction process will be triggered.

5.4.2.1 Contribution of PROV-TE

In order to audit the cause of Increase in Resource Request, PROV-TE model in Figure 5.6 specifically identifies the related attributes to the cause in hand. Following the dataflow when Activity: Evict Task is triggered by an increase in resources request (CPU and RAM), attributes to be investigated are identified, which are TEv_resource_request_CPU, and TEv_resource_request_RAM. These two entities are generated by Activity: Update Running Tasks. When a task is updated, its Entity: TEv_eventtype is updated as well. This entity helps in

identifying the updated task while running. The time for every update is also recorded by TEv_timestamp. By using the ME_MachineID Entity, the exact machines and their hosted tasks can be identified. ME_MachineID is used so that the associated tasks can be identified. Finally, by using TEv_timestamp ME_MachineID, and TEv_eventtype tasks that were evicted as a result of this cause have been identified. Algorithms 2a and 2b, tables 4.7 and 4.8 in Chapter 4, have been applied in this investigation.

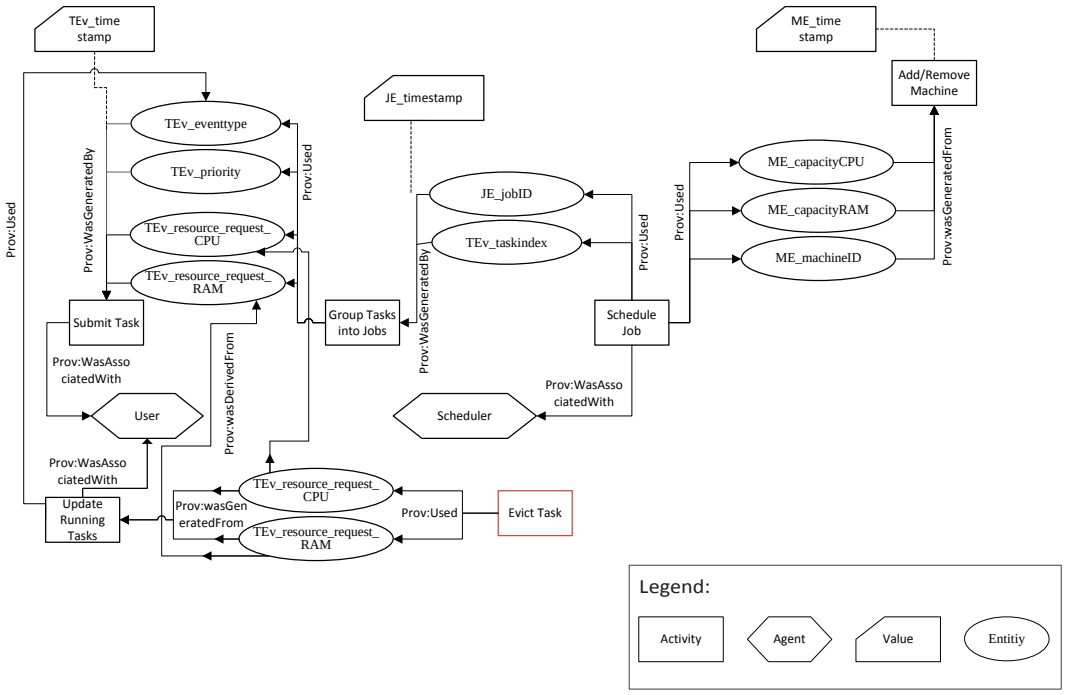

Figure 5.6 PROV-TE for Increase in Resources Request

5.4.2.2 Results and Analysis

It was found that 58,760 tasks requested more resources while running. Those tasks resulted in the eviction of other lower priority tasks. The timestamp of the resource update was used in order to count all evicted tasks within one interval from the update. The result shows that 1,583 tasks were evicted due to the increase in resources request by other higher priority tasks.

Users can request to update the resources' needed to compute their tasks. The update can be an increase of the original request or a decrease. In this investigation, tasks with an increase update have been identified and quantified. The increase in request sometimes result in an overload, where the sum of the needed resources to compute the hosted tasks exceeds the actual physical capacity. Thus, one or more tasks with lower priority get evicted as a result in order to make space for the resource request of the running higher priority tasks to be approved. It has been found in this investigation that 1,583 evicted tasks were linked to Cause 2 Increase In Resource Requests.

5.4.3 Investigation 3 (Evicted Tasks caused by Demand Exceeding Physical Capacities)

Usage threshold level is one of the measures which is used to trigger Task Eviction mitigating strategy in Cloud dataset centres to avoid machine overloads. Due to the limitations of the dataset with regards to usage threshold level not being documented, overload has been calculated by comparing the physical capacity against the requested resources on a daily basis as explained in Section 4.8.3. Every task that has been evicted in the timeframe of overload (requested capacity exceeds physical resources) is counted in this investigation.

5.4.3.1 Contribution of PROV-TE

From Figure 5.7, PROV-TE helps to identify entities related to requested resources and machines' capacities. Following the dataflow of Activity: Evict Task, entities like ME_capacityRAM, ME_capacityCPU, TEv_resource_request_CPU, TEv_resource_request_RAM are used. Also, by using TEv_taskindex and JE_JobID, ME_machineID, it has been possible to sum

the requested resources for all tasks per machine per day. Algorithms 3a and 3b, tables 4.9 and 4.10 in Chapter 4, have been applied in this investigation.

5.4.3.2 Results and Analysis

By comparing the results of the two algorithms 3a-b, it has been identified that days 1, 2, 9 and 10 were highly likely to have overload in both CPU and Memory

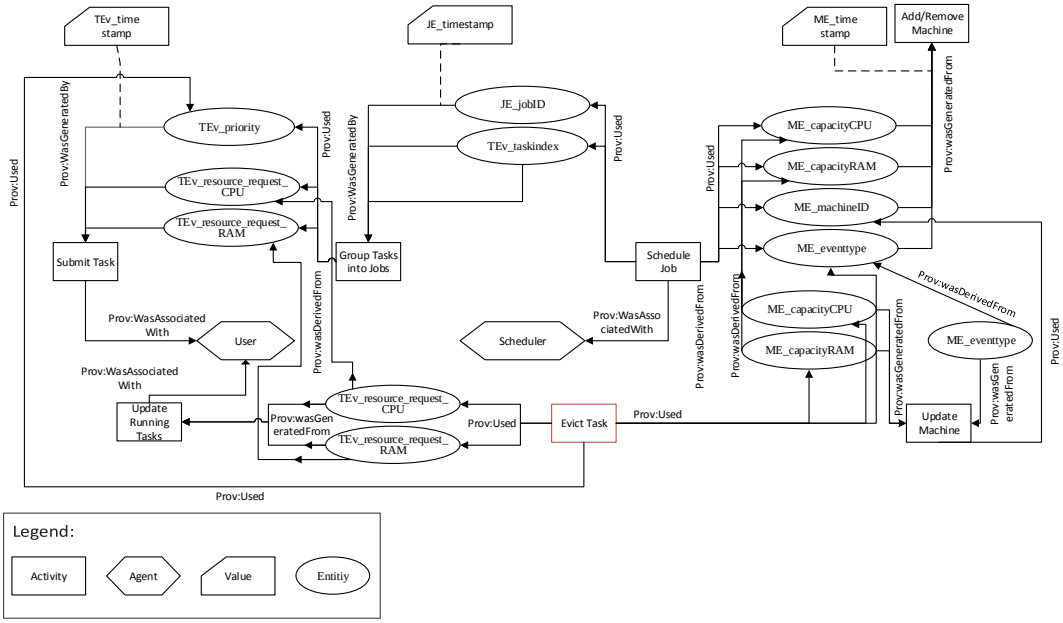

Figure 5.7 PROV-TE for Demand Exceeding Machines' Capacities

because the sum of the requested resources per day exceeds the sum of the actual physical capacity due to over-commitment as seen in Figures 5.8 and 5.9. There were 463,544 evicted tasks in these four days which are believed to be linked to Cause 3 Demand Exceeding Physical Capacities.

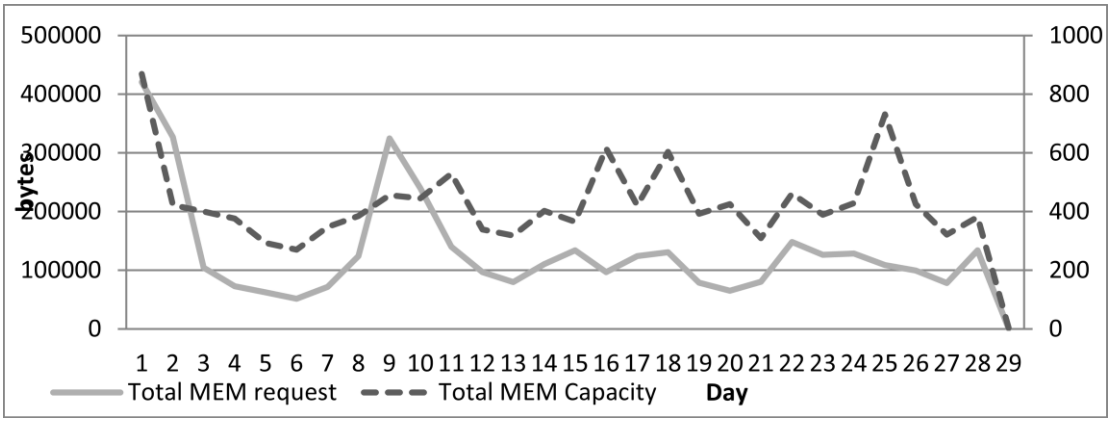

Figure 5.8 Total Memory Request vs Total Memory Capacity Over the 29-Day Dataset

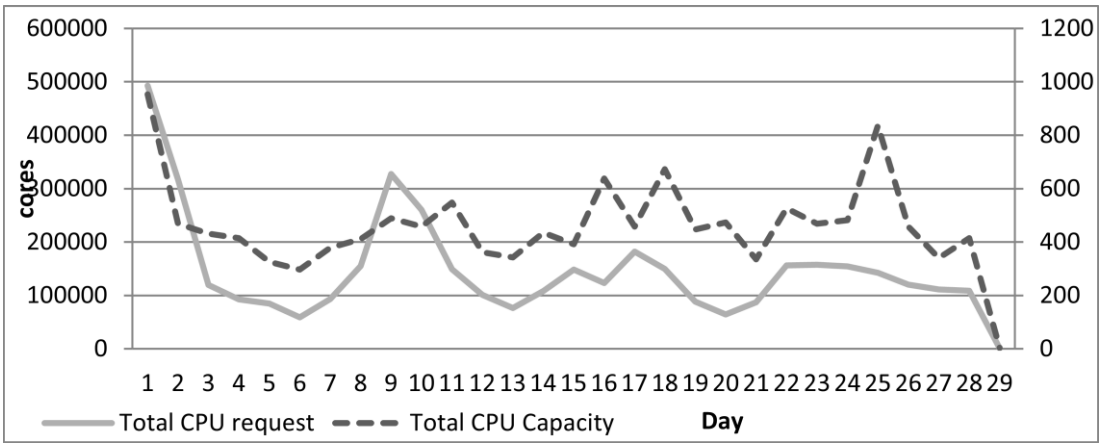

Figure 5.9 Total CPU Request vs Total CPU Capacity Over the 29-Day Dataset

5.4.4 Investigation 4 (Evicted Tasks caused by Missing Machines)

One of the causes that triggers Task Eviction is missing machines. Removal of physical machines can usually be caused due to maintenance or failure [7]. The attribute ME_eventtype of PROV-TE tells whether the machine is Added, Updated, or Removed. Tasks that have been scheduled to run on machines which are removed get evicted as a result. This investigation looks at the extent of this cause.

5.4.4.1 Contribution of PROV-TE

The investigation started with two entities, ME_eventtype and ME_timestamp shown in Figure 5.10. Machines, that were removed, have been identified and counted. To find the tasks that were evicted as a result, the following entities have been used: TEv_timestamp, TEv_eventtype, JE_JobID, and TEv_taskindex. Algorithms 4a and 4b, tables 4.11 and 4.12 in Chapter 4, have applied in this investigation.

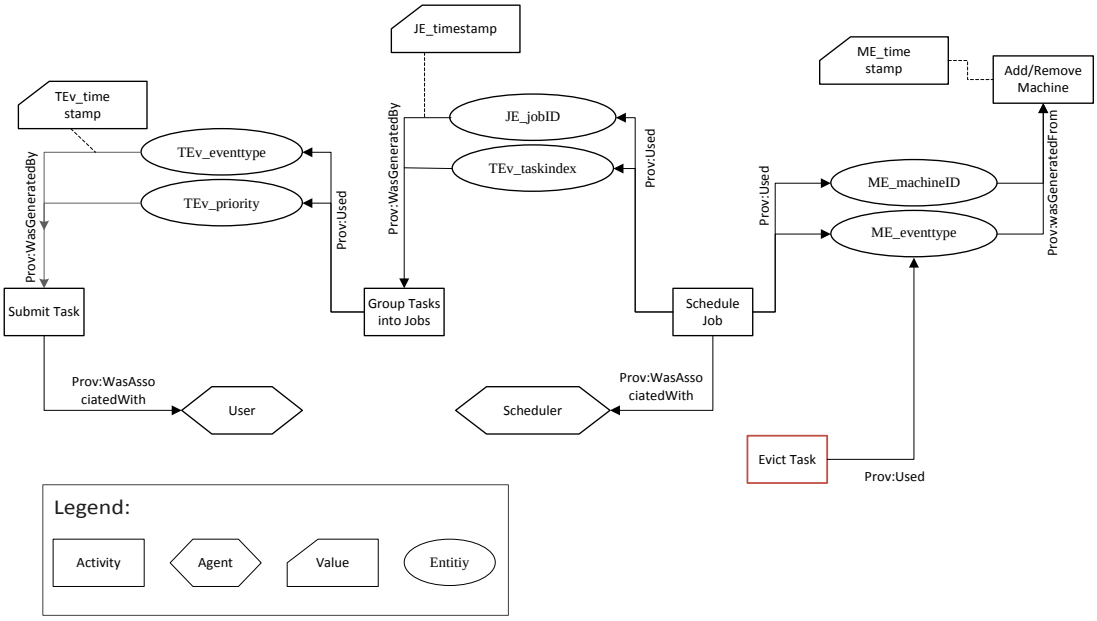

Figure 5.10 PROV-TE for Missing Machines

5.4.4.2 Results and Analysis

The diagnostic algorithms show that 5141 physical machines were removed during the period of the 29-day. 76 tasks found to were evicted within one interval from the removal of these machines. This cause is not considered to be dominant because it is linked to a low number of tasks were evicted.

5.4.5 Investigation 5 (Evicted Tasks caused by Decrease in Physical Machines Capacities)

Due to failure of nodes and maintenance, machines' capacities (CPU and Memory) can get reduced [5], [7]. That reduction can cause tasks to be evicted due to the lack of available resources to compute those tasks. The focus of this investigations is to identify those evicted tasks.

5.4.5.1 Contribution of PROV-TE

The following explains how the evicted tasks that are linked to *Cause 5 Decrease in Machine Capacity* can be identified by using PROV-TE. The process starts with the Activity: Evict Task. Tracing backwards in the dataflow, the needed attributes for this investigation are identified, shown in Figure 5.11. Specifically, the process checks the Entities: ME_capacityRAM and ME_capacityCPU which have had an update. The Entity: ME_machineID helps in terms of identifying the exact machines with a decrease in Memory or CPU and the IDs of the tasks that

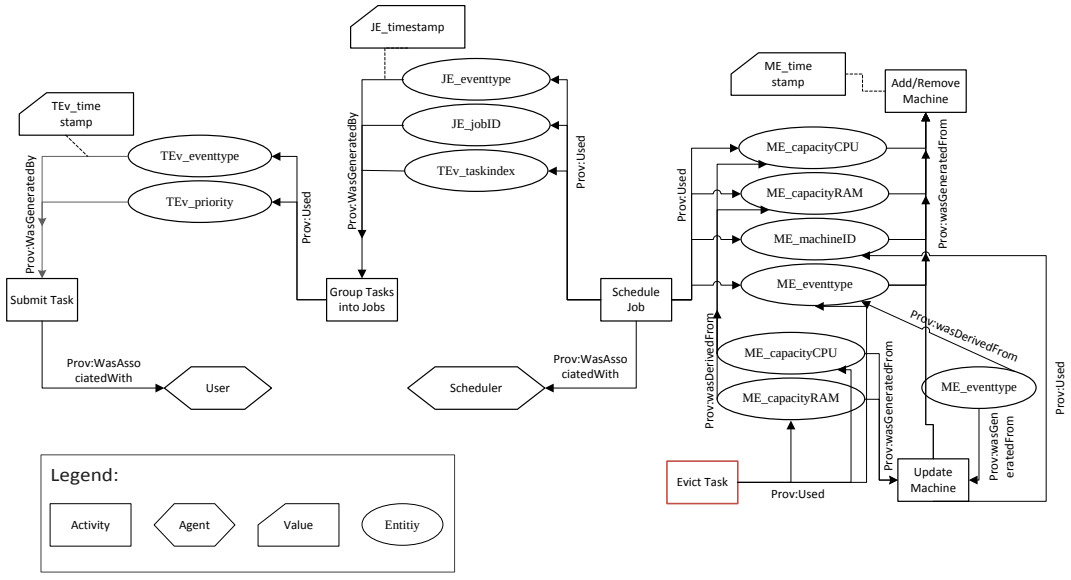

Figure 5.11 PROV-TE for Decrease in Machine Capacity

were scheduled on them by using JE_jobID, TEv_taskindex through the Activity: Schedule Job. Then, by using Entity: TEv_eventtype, it has become possible to identify the evicted tasks hosted on the specified machines earlier. Algorithms 5a and 5b, tables 4.13 and 4.14 in Chapter 4, have been applied in this investigation.

5.4.5.2 Results and Analysis

From Algorithm 5a: by comparing the initial capacity of the machine in the event type Add and Update, it was possible to narrow down the search to only the machines with a decrease in resources. During the 29-day period, it was found that 1267 physical machines have encountered an update (either increased or decreased) in their capacity. Only 32 (2.5%) of which have had their capacity decreased.

From Algorithm 5b, it has been found that 7,670 tasks were scheduled in the 32 machines and have been evicted within one interval from the update. Since Google logs the data every 5-min, an assumption has been made that any evicted task within this timeframe from the update is linked to the decrease of machine capacity. Table 5.2 shows the 32 machines with the number of evicted tasks as a result of their decrease in capacity (CPU and/or Memory). For example, machine with the ID number ‘317488637’ had its memory decreased from 0.749 bytes to 0.4995 bytes at time *t*. As a result of this decrease, 321 tasks were evicted at time *t+1*.

Table 5.2 Number of Evicted Tasks per Machine (with Decreased Capacity)

Machine ID	Number of Evicted Tasks	Machine ID	Number of Evicted Tasks	Machine ID	Number of Evicted Tasks
317488637	321	6315250734	187	6400066596	113
6264344062	286	6316827871	404	6401302061	102
6274355716	625	6322213339	415	6402941427	41
6280643141	376	6335261139	324	6408086842	228
6282149131	349	6344084916	220	6415978528	126
6285257156	285	6370662053	140	6415979192	37
6289355687	569	6390664602	48	6437385645	71
6289704471	843	6391270721	207	6453653899	88
6296268057	306	6391293459	209	6455072430	30
6296865278	256	6391374318	151	6457070948	54
6301942525	301	6391421427	125		

There is a correlation between the machines capacity reduced and the eviction of tasks. This could be due to the fact that the needed capacity to run the tasks exceeded the actual capacity after resources update. The result of this investigation shows that the Cause 5 Decrease in Machine Capacity is linked to 7,670 evicted tasks during the total period of the dataset. From such analysis, Cloud providers can know the machines that have abnormal behaviour or cause the most disturbance in the data centre by identifying the linked number of evicted tasks and time of evictions.

5.5 Overall Analysis

The overall findings show that PROV-TE was useful, due to the reasoning support it added, in identifying the dominant cause that triggered task eviction strategy, which is “Higher priority tasks take place over lower priority tasks” as

shown in Table 5.3. The reasons being the edges between the nodes which explain the relationships between the attributes. In order to tackle the dominant cause of overload, more understanding is needed and further investigation on the root causes could be conducted.

Table 5.3 Overall Findings of the Investigations

Cause **Ordered by Level of Impact**	**Number of Evicted Tasks**
Cause 1. Take Over by Higher Priority Tasks	1,421,054
Cause 3. Actual Demand Exceeds the Physical Capacity	463,544
Cause 2. Decrease in Physical Machine Capacity	7,670
Cause 5. Increase in Resource Requests	1,583
Cause 4. Missing Machines	76
Total Number of Evicted Tasks Found by the Use of PROV-TE	**1,893,927**

The total number of evicted tasks found by provenance-driven diagnostic framework is 1,893,827, which exceeds the number stated by the dataset by 33%, 1,422,317.

Certain information in the dataset has been obfuscated for confidentiality reason. This limits the ability of the diagnostic algorithms which might have affected the level of accuracy. For example, level of overload threshold is not stated. Due to this, a workaround process has been conducted to identify the evicted tasks linked to Cause 3 Demand Exceed Physical Capacity. In section 5.4.3, the method of identifying the extent of the Cause 3 was by calculating the sum capacity of physical resources and the sum requested resources. The

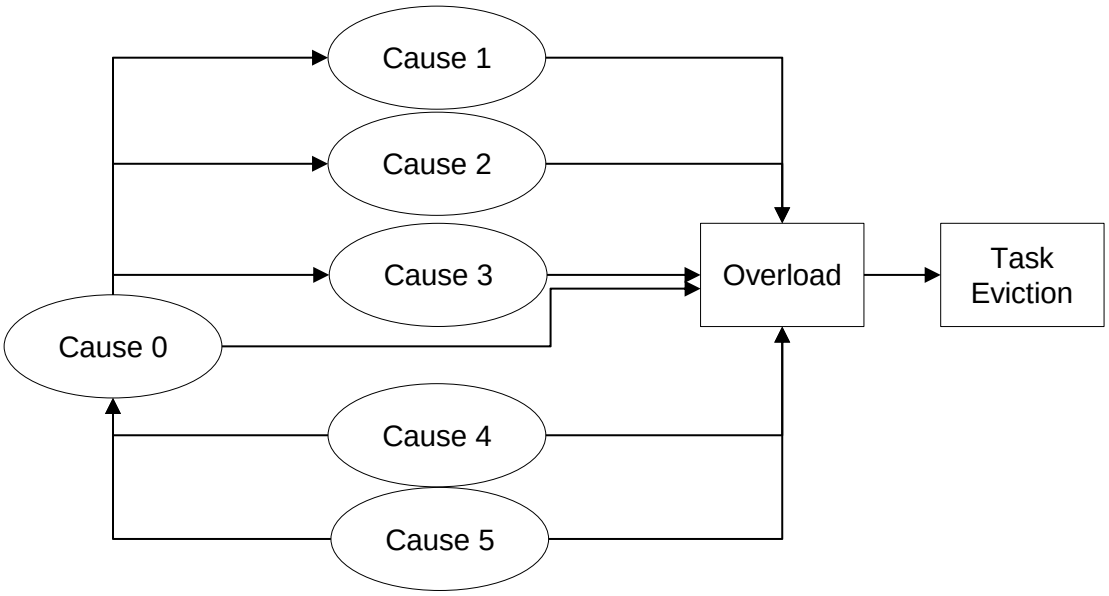

Figure 5.12 Chain of Causes

assumption is whenever the sum of requested capacity exceeds the physical capacity, one or more tasks could be evicted. Another particular challenge related to the identification of causes is the overwhelming volume and size of the dataset. Althought query performance is not a focus in this research, it was an obstacle. Thus, the level of granularity has been changed in terms of selecting only the needed attributes in order to be queried which increased the query performance, refer to section 3.3.1.

On a closer analysis, a cause can be the outcome of another cause, hence a possible cascade of causes. Not enough physical or virtual resources (Cause 0) is another cause that can lead to the five causes mentioned in section 4.3. In addition, each one of the six causes can itself be a root cause of another issue that leads to an overload. The chain of causes in the Google case study all lead to overload and consequently trigger Task Eviction overload mitigating strategy. Figure 5.12 illustrates the possible relationships between the six causes.

From Figure 5.12, it can be observed that Cause 0 can itself lead to overload but it can also lead to Cause 1 (Take Over of Higher Priority Tasks), Cause 2 (Increase in Resource Requests), and Cause 3 (Demand Exceeds Physical Capacity). Cause 4 (Missing Machines) and Cause 5 (Decrease in Machine Capacities) can also lead to both overload and to Cause 0. For example, Cause 4 can result in Cause 0 which later can lead to Cause 3. In this case, Cause 3 creates the overload instance where tasks are then evicted as a result. In this example, from the evicted tasks' point of view, the cause of the eviction is Cause 3 but the root cause is Cause 4. The figure represent a two-level cause; however, it is not a deterministic figure and needs further investigation.

Having said that, from Table 5.3, Cause 1 itself may not be the most dominant cause due to the fact that other causes could have contributed as well. Thus, a deeper investigation and analysis is needed to identify the root causes of every evicted task, which is a limitation of this study. PROV-TE could be used for the deeper-analysis investigation by guiding the development of more diagnostic algorithms following similar development logic presented in Chapter 3. The focus would be looking at the relationships between the five causes, one becoming the cause and the other becoming the effect and so on. The investigation would analyse the dataset to find evidence if in fact one cause has led to the existence of other cause(s). The outcome is envisioned to be a complete trace of the events and cause(s) that led to the eviction of every task.

Often in semantics the understanding of the dataset that underpin that construction of provenance models could have limitations but not the dataset in

question. In general, semantics has shortcomings, such as missing links and concepts, which limit its power to result in accurate answers or to answer more complex question [141].

To further examine the reasons behind the gap between the results of the investigations and the fact stated by the dataset, the Provenance-Driven Diagnostic Framework should be evaluated on another dataset. Next Chapter presents the use of a simulation tool to generate 15 heterogeneous datasets based on a hybrid configuration combining both Google and Amazon EC2 Cloud configurations to test and evaluate the framework. Applying the algorithms in a controlled environment could potentially lead to an opportunity of enhancement which could have a positive effect on accuracy.

5.6 Summary

This Chapter has presented and illustrated the use of Auditor for acquiring a deeper understanding of the causes of Task Eviction. It started by giving the context of the experiment and the purpose of this Chapter. It then described the aim and hypothesis of the experiment. Then it presented the application of the diagnostic algorithms in five clearly distinguished investigations, each with a unique focus. It showed the results obtained from the application of the diagnostic algorithms which quantifies the extent of every cause on the dataset. It also explained how PROV-TE contributes specifically to every cause and guides the investigations. It ended with identifying the most dominant and least dominant causes for Task Eviction. Also, it discussed the limitations of the given Cloud dataset and the probable effects on the diagnosis.

Chapter 6

Evaluation of the Diagnostic Algorithms

6.1 Introduction

This chapter presents the evaluation step of the Provenance-Driven Diagnostic Framework which assesses its accuracy of the identified evicted tasks and the causes that led to the eviction of those tasks. It describes the method used to evaluate the framework which is simulation. It gives an overview of the chosen simulation tool. It gives details on the general and scenario-specific simulation setup and configuration. The Chapter then shows the output of the simulation which is 15 heterogeneous and randomly-generated Cloud datasets based on a hypothetical configuration that combines Google's and Amazon's EC2 Clouds' configurations reflecting a general setup. The generated datasets are used for the evaluation of the framework. The Chapter then explains the accuracy of the framework by analysing the results based on the precision and recall statistical measures. Finally, it concludes with a summary of the findings.

6.2 Purpose and Scope of Evaluation

In Chapter 4, the Provenance-Driven Diagnostic Framework, which includes the developed diagnostic algorithms and the underpinning PROV-TE model, has been applied on Google 29-day Cloud usage dataset. Evicted tasks have been identified as well as the relevant causes based on metrics such as timestamp and shared physical machine. In order to evaluate and assess the framework, a simulation tool, SEED [14], has been used to generate Cloud datasets according to known Task Eviction behaviours.

The hypothesis of the research is that PROV-TE adds value to the raw data by connecting the data in a way that provides additional meaning for further interpretation and analysis. Specifically, the analysis will provide the reasons and causes of an overload.

The aim of this evaluation is to further test and evaluate the reasoning power of the proposed diagnostic algorithms and the underpinning PROV-TE model for the different overload scenarios. Due to the limited access to real Cloud datasets, the simulation tool has been set up with a general data centre configuration and has been used to generate 15 different simulated datasets. Each dataset comes with a log which includes details of the physical and virtual machines such as Host ID and CPU/MEM units, and tasks such as requested units of CPU/MEM and priority. Most importantly, it includes details about eviction of tasks such as relevant eviction cause and timestamp. These details will be used to validate the results of our framework by calculating the precision and recall of every diagnostic algorithm. Also, having datasets that reflect general data centre configuration from different Cloud vendors will illustrate the transferability of the diagnostic algorithms for task evictions driven by PROV-TE model. The applicability of working with different datasets generated from different configurations will be shown.

The experiment will focus on the following causes which could potentially trigger overloading; hence Task Eviction:

Cause 1. *Take Over by a Higher Priority Task* - higher priority tasks will always be scheduled irrespective of remaining machine capacity.

Cause 2. *Increase in Resource Requests* by a Running Task – each physical machine has a fixed capacity. During the execution of a task, occasionally

a task could request more resources. If the task has a higher priority, the request will be approved regardless of the remaining machine capacity.

Cause 3. *Actual Demand Exceeding Physical Capacities* – when over-commitment is applied, more virtual resources are allocated than the actual physical capacities. At some point in time, users could use all of their allocated resources. At this stage, one or more tasks might not be computed due to the degradation in physical resources. In this case, overload occurs when the maximum physical usage exceeds the threshold usage level of the machine's capacity.

The scope of the evaluation focuses in these three causes so that a reasonable range of typical patterns of behaviour in resource management at the IaaS level of Cloud computing is covered.

6.3 Simulation Tool

The Simulation EnvironmEnt Distributor (SEED) tool [14] has been used to systematically generate different Cloud datasets, each with a different task eviction behaviour. Simulation in computer science domain is a vital systematic method for validating complex behaviours. In Cloud environments, simulation is the favoured method for evaluation due to the dynamic conditions of Cloud data centres and their scale [13], [142]. Evaluating new mechanisms and frameworks in a randomized, repeatable, reliable, isolatable and scalable manner can be achieved through the use of simulation [81], [143]–[145]. Simulation also allows the abstraction of system complexities which makes it possible to focus on specific variables under controlled and configurable parameters permitting the repetition of experiments and unbiased comparison of results.

6.3.1 Simulation Tools for Clouds

In Cloud computing domain, there is a limited number of simulation tools that can be used [146]. They share common features but every tool has unique characteristics and focus. For instance, GreenCloud [147], YANS [148] focus on Cloud network parameters. Haizea [149] focus on scheduler performance. MDCSim [150] is only for commercial access and does not consider virtualization and multiple tenants. Other simulation tools such as Cloudsim [13], iCanCloud [151], DCSim [152] and SEED [14] are more generic. Table 6.1 shows the analysis of different simulation tools. SEED has the following advantages:

1. *Event-based synchronization is supported while maintaining reasonable levels of performance* compared to real world time. Performance is an important metric for users and administrators in terms of business requirements and operational costs such as money and time.
2. *Setup is guided and requires minimal user intervention and expertise in terms of configuration and programming.* This feature permits rapid development and execution of simulation. Also, it enables SEED to be provided as a SaaS.
3. *Assumptions about the underlying hardware of the simulator are not essential to execute distributed simulations.* Simulations can run across heterogeneous machine architectures and operating systems.
4. *Low-level understanding of both the model domain as well as aspects relating to simulation synchronization is not a prerequisite*. SEED facilitates the modelling of the domain based on graph notation and was designed specifically for modelling large-scale data centres.

Table 6.1 Comparison of Cloud computing Simulation Tools

Simulation Tool / Focus Domain	Model Elements	VM Support	Accessibility
YANS [148] / Network	Task, Consumer, Scheduler, Network,	No	Open Source
Haizea [149] / Scheduler	Task, Server, Scheduler	Yes	Open Source
GreenCloud [147] / Network	Task, Server, Scheduler, Network	No	Open Source
MDCSim [150] / Multi-tier system	Task, Consumer, Server, Scheduler, Data Centre, Network	No	Commercial
Cloudsim [13] / Environment	Task, Consumer, Server, Data Centre, Scheduler, Network	Yes	Open Source
iCanCloud [151] / Environment	Task, Consumer, Server, Data Centre, Scheduler, Network	Yes	Commercial
DCSim [152] / Environment	Task, Server, Data Centre, Scheduler	Yes	Commercial
SEED [14] / Environment	Task, Consumer, Server, Data Centre, Scheduler, Network	Yes	Commercial

SEED has been selected for this research mainly due to points 2 – 4 above. In addition, it is a product from our research group; hence access is allowed [14].

6.3.2 Overview of SEED

In [14], Cyber-Physical Systems (CPS) simulation can be performed by the assembly of SEED's core components which are formed by several services. Its high level architecture, is shown in Figure 6.1. SEED's architecture is loosely-coupled. The components are less dependent on each other. Performance bottlenecks are reduced because simulation components are located on heterogeneous machines within a network. SEED allows the addition of further components into the system. The characterization of the important components are as follows:

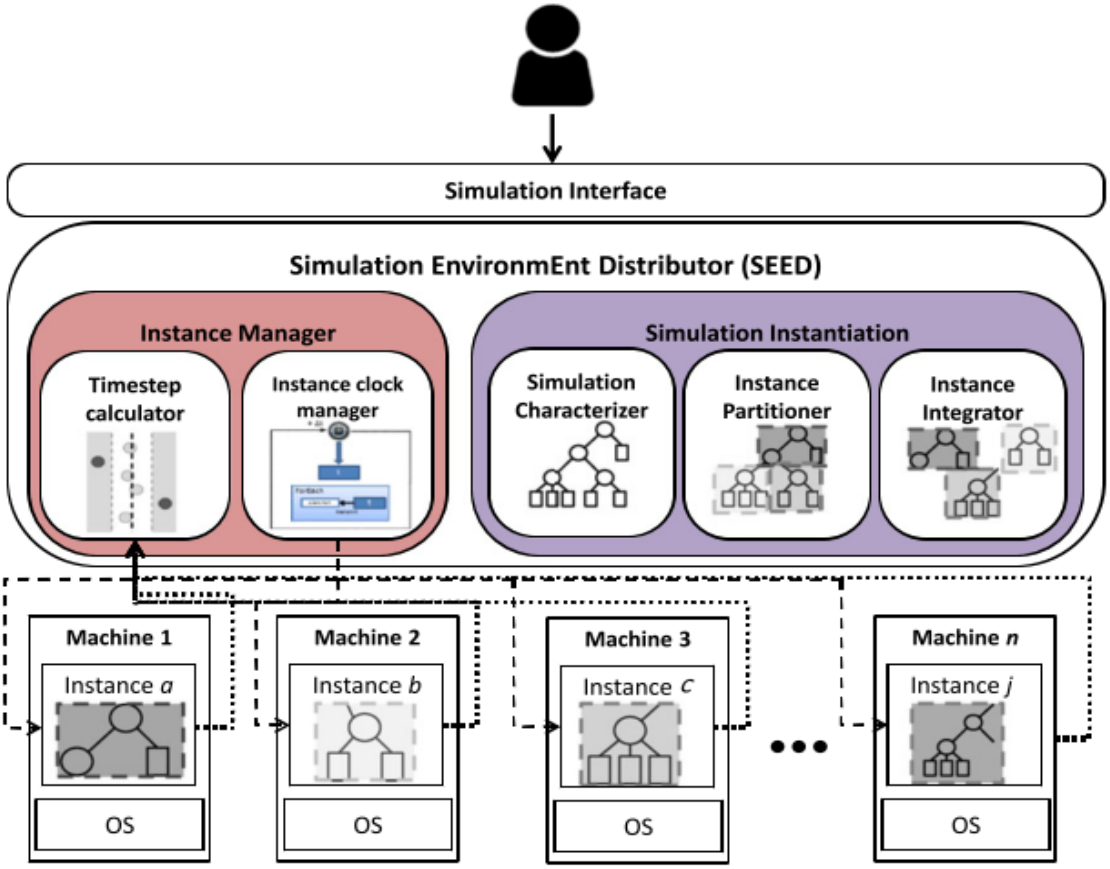

Figure 6.1 SEED High Level Architecture [14]

- *Simulation Instantiation*: the creation and classification of the simulated network topology in addition to the configuration and partitioning of a simulation across distributed infrastructure are automated.
- *Instance Manager*: Highly synchronized simulation that is deployed across distributed infrastructure is maintained by the provided scalable approach. Message ordering between virtual nodes that exist within different partitions is managed by the provided open synchronization framework between local clocks for instances. Clock Manager manages the simulation local clock of every instance with respect to the global clock of the entire simulation.

- *Instances*: Nodes, Links, and Tasks are interacting components that form a logical unit of simulation computation. Each instance, which is formed by a specified set of tasks and a subset of the total virtual network topology, executes on a partition that is hosted on a unique physical machine and automatically created by SEED. The Clock Manager externally manages each instance's local clock.

6.4 Simulation Design

As illustrated in Table 6.2, three scenarios have been developed in SEED. Scenario 1 includes the behaviour of Cause 1. Scenario 2 includes the behaviours of Cause 1 and Cause 2. Lastly, Scenario 3 includes the behaviours of Cause 1 and Cause 3.

Following the scope of this evaluation mentioned in section 6.2, in scenario 1, one or more lower tasks are expected to be evicted when a higher priority task is to be scheduled and there is lack of available resources. In scenario 2, one or more lower priority task are expected to be evicted when there is a lack of free resources and when (1) a higher priority task is to be scheduled, or (2) a higher priority task requests more resources at runtime. In scenario 3, one or more lower priority tasks are expected to be evicted when there is a lack of free resources

Table 6.2 Design of Scenarios

	Scenario 1	**Scenario 2**	**Scenario 3**
Cause (C1)	•	•	•
Cause (C2)		•	
Cause (C3)			•

and when (1) a higher priority task is to be scheduled, or (2) the actual demand exceeds the actual physical capacity controlled by the overload threshold level.

6.4.1 General Setup for the Simulation Environment

The simulation environment has been configured to reflect a general data centre setup and can be seen in Figure 6.2. For every run, the tool starts with building 20 physical machines (PM) and 40 virtual machines (VM). Each PM has two VMs (1:2). The PMs' CPU and RAM sizes are fixed with 8 units and 15 GB, respectively. The VM sizes are chosen randomly from a specified size list. Number of VM CPUs can be 2, 4, or 8 units. VM RAM size can be 4, 6, or 8 GB. The sizes are a reflection of Amazon EC2 c3.2xlarge instance [153]. However, VM and PM sizes of other vendors can be used. This particular instance allows over-commitment of resources. The scale of the simulation can be generalized to larger environments with more PMs and VMs, generating huge volumes of data. This aims to demonstrate the feasibility of massive-scale simulation for implementing provenance-based techniques.

Tasks are then generated according to a random task submission rate (TSR). TSR is randomly chosen from 100-300 per hour. The simulation length is 24 hours. The method of task distribution is: send one task to one VM at a time, in equal distribution, then loop back again until all tasks are sent to be queued in every VM.

There are 4 variables assigned to each task. Firstly, a task's length is measured in steps. The task length is randomly chosen from 2 to 10 steps. The length of the task is the number of steps needed to finish execution. In SEED, events are logged in a one-step interval. A step is a predefined interval of 30 seconds.

Second, a priority is randomly assigned to each task. It is a number to define the privilege of a task: 0 (lowest), 1 and 2 (highest). Finally, the remaining two variables are the requested resources, CPU and RAM. The resources are also chosen randomly from a predefined list (1, 2, 3, 4, 5, 6, 7, 8).

For every VM, there are three queues for tasks to be scheduled, once for each priority which ensures that every task get its fair time of waiting in the queue. In the scheduling method, there is a loop that goes around the three queues and dequeues one task from each queue at a time to be scheduled.

6.4.2 Scenario's Specific Configuration

Each scenario has additional configuration in order to generate the needed behaviour in the dataset. Three algorithms are developed in SEED to mimic the behaviour predefined in the three scenarios, namely Task Evictor, Request

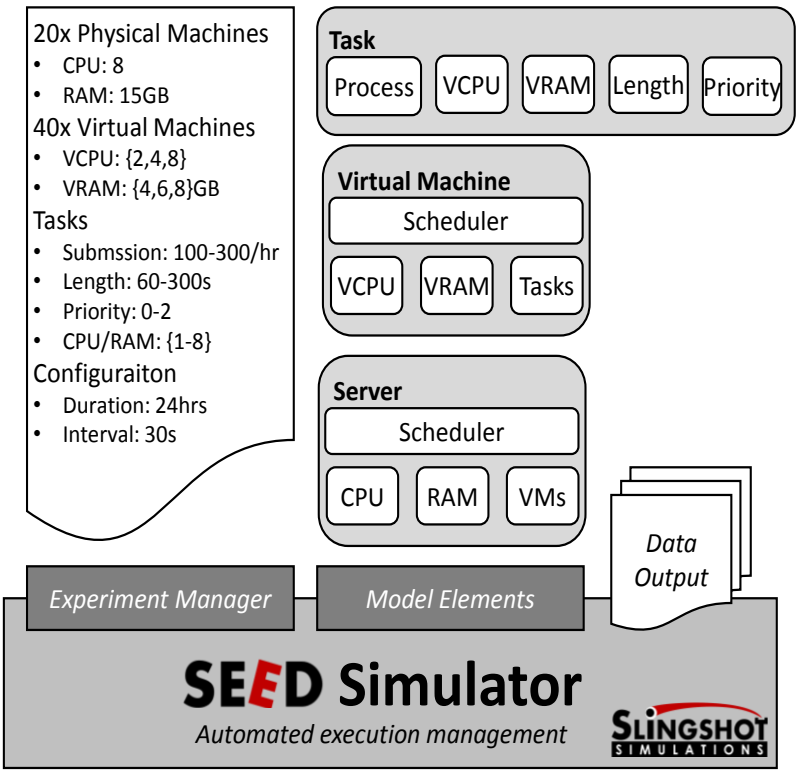

Figure 6.2 Configuration of the simulation environment using the SEED simulator

Handler and Overload Manager. A brief description of these is detailed below. In the case where only one loop is filled (equal-priority), tasks are then scheduled in a First-Come First-Served order, following the priority scheduling algorithm [154].

6.4.2.1 Cause 1: Take Over by Higher Priority Tasks

For every VM, whenever there is a lack of RAM or CPU and there is a task waiting in the queue with higher priority than the ones running, lower priority tasks get evicted so the VM to be ready to schedule the waiting higher priority task.

Table 6.3 Algorithm: SEED Task Evictor

```
    Task A is to be hosted in VM n
1. IF available resources on VM n <= Task A requested CPU/RAM
2.         SCHEDULE Task A
3. ELSE IF there are lower priority tasks than Task A
4.         CREATE a List of tasks where their priority <  priority of Task A.
5.         ORDER elements of List in ASCENDING order by priority.
6.         IF SUM of resource of List >= Task A requested Resource
7.             SET TotalFreedResource = 0
8.             WHILE (TotalFreedResource < Requested Resource)
9.                 TotalFreedResource += Task Recourse
10.                KILL Task in List
11.            END WHILE
12.            SCHEDULE Task A
13.        ELSE
14.            WAIT in queue to be scheduled
15.ELSE
16.        WAIT in queue to be scheduled
```

As illustrated in Table 6.3, whenever a task is to be scheduled, the scheduler has been configured to first check the available VM capacity, CPU and RAM. If there is enough space, then the task gets scheduled. Otherwise, if there lower priority tasks running in the VM, a list is then created to include all lower priority tasks ordered ascendingly by priority. From the top of the list, tasks get evicted until enough space becomes available. Then the task in question gets scheduled. In

case there are no lower priority tasks, the to-be-scheduled task is to wait in the queue until free space becomes available. Because all behaviours share the same policy of evicting tasks, SEED has been designed to use the algorithm SEED Task Evictor for every scenario.

6.4.2.2 Cause 2: Increase in Resource Request

Scheduling a task on a specific VM depends on the task's requested capacities, in terms of CPU and RAM. Once hosted, a task can request a change in the requested resources. In case there is no free resources to accommodate this request and there are lower priority tasks on the same VM, the task eviction mechanism will be executed until the desired requested capacities become available.

Table 6.4 Algorithm: SEED Request Handler

```
Task A is to be hosted in VM n
1.  WHILE Task is in progress < 1  // 1 = finished
2.          Generate random request
3.          IF (Task A new request < VM's CPU/RAM && there is free
            space)
4.                  Approve request
5.          ELSE IF there are lower priority tasks
6.                  RUN Algorithm: SEED Task Evictor to evict lower priority
                    tasks
7.                  Approve request
8.          ELSE
9.                  Deny request
10. END WHILE
```

As illustrated in Table 6.4 while a task is running, the simulation tool has been configured to generate a new random request from a predefined list (1, 2, 3, 4, 5, 6, 7, 8). The request will get approved only if there is available space in the VM. Otherwise, the same lower priority task eviction method of Algorithm 4: Task Evictor is run until the requested space becomes available.

6.4.2.3 Cause 3: Demand Exceeds the Physical Capacities

Over-commitment is a policy that is widely adopted in data canters to maximize resources' usage. Physical and virtual usage are managed by overload threshold levels [155]. The simulation tool has been configured with an 80% physical threshold usage level. Once physical usage exceeds it, eviction process is executed. Unlike the other scenarios, two policies are enforced when evicting tasks, lower priority task first and Last-In-First-Out (LIFO). It is more sensible to evict tasks that have just started than those near to finish.

Table 6.5 Algorithm: SEED Overload Manager

Every PM has 2 VMs
Every VM has more than 1 task
1. **SET** threshold level
2. **SUM total** PM usage
3. **Calculate** the usage ratio based on the threshold level
4. **IF** PM total usage > threshold
5. **CREATE** a List of tasks running in the PM
6. **ORDER** elements of List in ASCENDING order by priority
7. **ORDER** elements of List in DESCENDING order by Time of hosting in VM
8. **WHILE** (PM total usage > threshold)
9. **RUN** Algorithm: Task Evictor to evict lower priority tasks
10. **END WHILE**
11. **END IF**

As illustrated in Table 6.5, for every physical machine, the total physical usage is calculated every one-step interval (30 seconds). If the total physical usage exceeds the predifented usage thresold limit, tasks get evicted following the same lower priority task eviction method of SEED Task Evictor until normal usage behaviour is restored. The list is ordered ascending by tasks priority and descending by time of hosting in the VM. In this controlled environment, normal usage behaviour means the total physical usage is less than the usage thresold limit.

6.5 Simulation Runs

Each scenario (simulation) is run 5 times, each resulting in a dataset (trace log) similar to the log data from a data centre. Having a dataset with more than one Cause will help validate the accuracy of the diagnostic algorithms. These datasets will be then be cleansed and imported into a database to be diagnosed and analysed by the diagnostic algorithms.

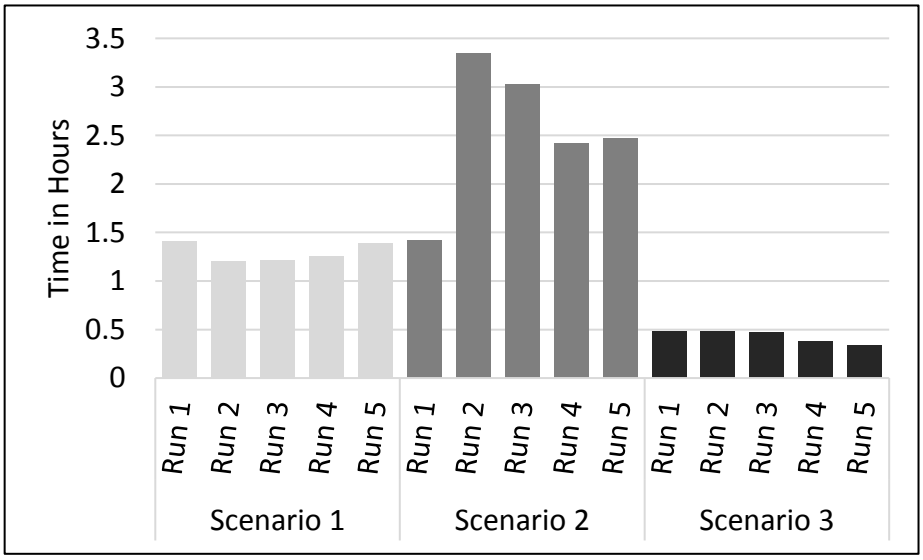

Figure 6.3 Execution time for every simulation run

Figure 6.3 shows the execution time of every run for every scenario. The mean execution time for all runs is 1.29 hours for scenario 1, 2.53 hours for scenario 2, and 0.43 hours for scenario 3. The standard deviation of the mean execution times is ±0.1 for scenario 1, ±0.73 for scenario 2, and ±0.06 for scenario 3. Due to the small number of tasks of Scenario 3 which depends on the random TSR, the mean of Scenario 3 execution times is smaller compared to the means of Scenarios 1 and 2. Also, Scenario 2 execution times are relatively higher is because of the complexity of Algorithm: Request Handler.

While tasks are running, their resources request are constantly and randomly changed. In order for every request to be approved or not, capacities comparison is undertaken. Every request in RAM or CPU must be less or equal the size of the VM. Also, if there is no VM space and there are lower priority tasks, the eviction process is triggered. Only then the request is approved. This explains the long execution times for runs of scenario 2.

6.6 Simulation Output

To show the randomness, variances and differences of the resulting generated behaviour, each scenario has 5 simulation runs. The output of SEED is a total of 15 simulated and randomly-generated logs. Table 6.6 summarizes the overall

Table 6.6 Output of Simulation

	Scenario 1		**Scenario 2**			**Scenario 3**		
	Total Tasks	***Total Evicted Tasks***	***Total Tasks***	***Total Evicted Tasks***		***Total Tasks***	***Total Evicted Tasks***	
Run 1	4208	259	3735	C1	C2	3131	C1	C3
				904	20		326	54
Run 2	4530	266	4076	C1	C2	2341	C1	C3
				967	20		16	187
Run 3	4501	421	4328	C1	C2	2687	C1	C3
				1041	45		1	176
Run 4	4653	297	4035	C1	C2	3077	C1	C3
				1012	37		177	76
Run 5	4538	319	4049	C1	C2	2596	C1	C3
				994	43		96	157

output of the 15 runs of the three scenarios. For each run, the total number of tasks as well as the total number of evicted tasks are shown. Also, because Scenario 2 and 3 have two causes each, the total number of evicted tasks related to each cause is also shown. The aim for Table 6.6 is that it will be used after applying the algorithms of the framework to calculate the precision and recall of the results, which will be explained in the next sections.

6.6.1 Simulation Parameters

Each simulation run results in one trace log (dataset) which combines all parameters of the simulation, physical machine, virtual machine, and task. The logs are in a csv format. There are shared parameters across all scenarios, explained in Table 6.7. As a result of the scenario-specific configurations mentioned in Section 6.4.2, there are unique parameters for scenarios 2 and 3, explained in Table 6.8.

Table 6.7 Common Simulation Parameters in all Trace Logs

Component	Parameter	Description
Simulation	SimTime	The simulation logging interval; step.
	UTCDateStamp	The date of the each logged interval
	UTCTimeStamp	The time of the each logged interval
	ID	The internal ID of every component
	ComponentType	The component type; PM, VM, Task
PM	PMemory	The physical Memory size of the PM
	PCPU	The physical CPU size of the PM
VM	VCPU	The virtual CPU size of the VM
	VMemory	The virtual memory size of the VM
	HostMachine	The ID of the Host PM
Task	Status	The status of a task; Not Started, Started, Executing, Finished, Killed (Evicted).
	HostID	The ID of the host VM.

	HostCPU	The CPU size of the host VM
	HostMemory	The Memory size of the host VM
	Length	The length of the task measured in SimTime (steps)
	Progress	The progress of the task; <1 = running, 1 = finished.
	Priority	The priority of the task
	CPU	The requested CPU size of the task
	Memory	The requested Memory size of the task

In Table 6.7, The parameters related to the *component: Simulation* and *parameter: Progress* in *component: Task* are mandatory for the working of the simulation tool but are not considered in the algorithms. VM-related parameters have been introduced due to the controlled environment which led to the enhancement of the algorithms, see section 6.7.1. The other parameters are not new for the proposed PROV-TE framework.

Table 6.8 Additional Scenario-Specific Parameters in Trace Logs for Scenario 2 and Scenario 3

Scenario	Component	Parameter	Description
Scenario 2	Task	isCPUChanged	A boolean variable to state if task requested new CPU size
		ReqCPU	The newly approved CPU size request
		isMemoryChanged	A Boolean variable to state if task requested new Memory size
		ReqMemory	The newly approved Memory size request
Scenario 3	Physical Machine	Overload	A Boolean variable to state if usage exceeded threshold in the PM
		PhysicalMemUsage	The total Memory usage of the PM
		OverloadMemThreshold	The predefined threshold Memory usage level

		PhysicalCPUUsage	The total CPU usage of the PM
		OverloadCPUThreshold	The predefined threshold CPU usage level

Depending on the scope of the running scenario, functions of other scenarios are turned off and the related parameters are not logged.

6.6.2 Simulation Logs

Below is a snapshot of one log of every scenario. Figures 6.4 - 6.6 show the datasets for scenarios 1, 2 and 3, respectively.

SimTime	UTCDateS	UTCTimeStam	ID	ComponentType	PMemory	PCPU	Status	VCPU	VMemory	HostMach	HostID	HostCPU	HostMem	Length	Progress	Priority	CPU	Memory
10	23-10-16	'21:00:01.2492	19	Aziz_Experiment.Components.Server	15	8	Executing											
10	23-10-16	'21:00:01.2492	77	Aziz_Experiment.Components.VirtualMachine			Executing	8	6	Aziz_Experiment.Components.Server : 19								
10	23-10-16	'21:00:01.2492	T_21_H_1	Aziz_Experiment.Components.Task			Started			Aziz_Exper	77	8	6	10	0	2	2	1
10	23-10-16	'21:00:01.2492	79	Aziz_Experiment.Components.VirtualMachine			Executing	8	8	Aziz_Experiment.Components.Server : 19								
10	23-10-16	'21:00:01.2492	T_22_H_1	Aziz_Experiment.Components.Task			NotStarted			Aziz_Exper	79	8	8	6	0	0	1	5
10	23-10-16	'21:00:01.2492	21	Aziz_Experiment.Components.Server	15	8	Executing											
10	23-10-16	'21:00:01.2492	81	Aziz_Experiment.Components.VirtualMachine			Executing	4	8	Aziz_Experiment.Components.Server : 21								
10	23-10-16	'21:00:01.2492	T_23_H_1	Aziz_Experiment.Components.Task			Started			Aziz_Exper	81	4	8	4	0	2	2	1
10	23-10-16	'21:00:01.2492	83	Aziz_Experiment.Components.VirtualMachine			Executing	4	6	Aziz_Experiment.Components.Server : 21								
10	23-10-16	'21:00:01.2492	23	Aziz_Experiment.Components.Server	15	8	Executing											
10	23-10-16	'21:00:01.2492	85	Aziz_Experiment.Components.VirtualMachine			Executing	8	8	Aziz_Experiment.Components.Server : 23								
10	23-10-16	'21:00:01.2492	T_24_H_1	Aziz_Experiment.Components.Task			Started			Aziz_Exper	85	8	8	10	0	2	5	5
10	23-10-16	'21:00:01.2492	87	Aziz_Experiment.Components.VirtualMachine			Executing	4	4	Aziz_Experiment.Components.Server : 23								
10	23-10-16	'21:00:01.2492	T_25_H_1	Aziz_Experiment.Components.Task			Started			Aziz_Exper	87	4	4	9	0	2	4	2

Figure 6.4 A snapshot of scenario 1 trace log

SimTime	UTCDateSt	UTCTimeSt	ID	Componer	Status	VCPU	VMemory	HostMach	HostID	HostCPU	HostMem	Length	Progress	Priority	CPU	isCPUCha	ReqCPU	Memory	isMEMCh	ReqMemory
8	25-08-16	'17:13:14.	T_1_H_1	Aziz_Exper	Executing			Aziz_Exper	41	4	6	4	0.25	0	1	FALSE	0	2	TRUE	4
8	25-08-16	'17:13:14.	43	Aziz_Exper	Executing	8	6	Aziz_Experiment.Components.Server : 1												
8	25-08-16	'17:13:14.	T_13_H_1	Aziz_Exper	Started			Aziz_Exper	43	8	6	2	0	1	7	FALSE	0	6	FALSE	0
8	25-08-16	'17:13:14.	3	Aziz_Exper	Executing															
8	25-08-16	'17:13:14.	45	Aziz_Exper	Executing	2	6	Aziz_Experiment.Components.Server : 3												
8	25-08-16	'17:13:14.	47	Aziz_Exper	Executing	2	4	Aziz_Experiment.Components.Server : 3												
8	25-08-16	'17:13:14.	5	Aziz_Exper	Executing															
8	25-08-16	'17:13:14.	49	Aziz_Exper	Executing	4	8	Aziz_Experiment.Components.Server : 5												
8	25-08-16	'17:13:14.	T_14_H_1	Aziz_Exper	Started			Aziz_Exper	49	4	8	8	0	2	3	FALSE	0	2	FALSE	0
8	25-08-16	'17:13:14.	T_2_H_1	Aziz_Exper	Killed				49	4	8	4	0	0	4	FALSE	0	7	FALSE	0
8	25-08-16	'17:13:14.	51	Aziz_Exper	Executing	4	6	Aziz_Experiment.Components.Server : 5												
8	25-08-16	'17:13:14.	T_15_H_1	Aziz_Exper	Started			Aziz_Exper	51	4	6	10	0	1	3	FALSE	0	1	FALSE	0

Figure 6.5 A snapshot of scenario 2 trace log

6.7 Output from the Diagnostic Algorithms

For the rest of the chapter, Cause 1 is also referred to as C1, Caused 2 is also referred to as C2 and Causes 3 is also referred to as C3. The following shows the results of the implementation of the diagnostic algorithms for Causes 1 - 3

SimTime	UTCDate	UTCTime	ID	Compon	Overload	PhysicalM	Overload	PhysicalC	Overload	PMemory	PCPU	Status	VCPU	VMemory	HostMac	HostID	HostCPU	HostMem	Length	Progress	Priority	CPU	Memory
11	10-10-16	'17:36:31.	77	Aziz_Experiment.Components.VirtualMachine								Executing	8	6	Aziz_Experiment.Components.Server : 19								
11	10-10-16	'17:36:31.	T_17_H_1	Aziz_Experiment.Components.Task								Finished			Aziz_Expe	77	8	6	3	1	2	5	2
11	10-10-16	'17:36:31.	79	Aziz_Experiment.Components.VirtualMachine								Executing	2	4	Aziz_Experiment.Components.Server : 19								
11	10-10-16	'17:36:31.	T_25_H_1	Aziz_Experiment.Components.Task								Executing			Aziz_Expe	79	2	4	5	0	1	1	1
11	10-10-16	'17:36:31.	21	Aziz_Expe	FALSE	0	0	0	0	15	8	Executing											
11	10-10-16	'17:36:31.	81	Aziz_Experiment.Components.VirtualMachine								Executing	4	4	Aziz_Experiment.Components.Server : 21								
11	10-10-16	'17:36:31.	83	Aziz_Experiment.Components.VirtualMachine								Executing	8	8	Aziz_Experiment.Components.Server : 21								
11	10-10-16	'17:36:31.	T_18_H_1	Aziz_Experiment.Components.Task								Executing			Aziz_Expe	83	8	8	7	0	1	6	6
11	10-10-16	'17:36:31.	23	Aziz_Expe	FALSE	0	0	0	0	15	8	Executing											
11	10-10-16	'17:36:31.	85	Aziz_Experiment.Components.VirtualMachine								Executing	8	4	Aziz_Experiment.Components.Server : 23								
11	10-10-16	'17:36:31.	T_27_H_1	Aziz_Experiment.Components.Task								Started			Aziz_Expe	85	8	4	4	0	1	3	4
11	10-10-16	'17:36:31.	87	Aziz_Experiment.Components.VirtualMachine								Executing	2	8	Aziz_Experiment.Components.Server : 23								
11	10-10-16	'17:36:31.	25	Aziz_Expe	TRUE	4	0	10	0	15	8	Executing											
11	10-10-16	'17:36:31.	91	Aziz_Experiment.Components.VirtualMachine								Executing	4	4	Aziz_Experiment.Components.Server : 25								
11	10-10-16	'17:36:31.	T_20_H_1	Aziz_Experiment.Components.Task								Executing			Aziz_Expe	91	4	4	5	0	2	4	3
12	10-10-16	'17:36:32.	T_21_H_1	Aziz_Experiment.Components.Task								Killed				55	8	8	4	0	1	8	7
12	10-10-16	'17:36:32.	9	Aziz_Expe	FALSE	3	0	1	0	15	8	Executing											

Figure 6.6 A snapshot of scenario 3 trace log

presented in Chapter 3, mainly the Auditor component. For scenario 1, the Auditor will only apply C1 related algorithms. For scenario 2, the Auditor will apply C1 and C2 related algorithms. For scenario 3, the apply will trigger C1 and C3 related algorithms, following the simulation design illustrated in Table 6.2.

The input to the Auditor was the 15 simulated datasets. The diagnostic algorithms were applied to find the causes of all evictions. The output of the Auditor is the identification of causes and relevant evicted tasks.

Table 6.9 summarizes the output of all diagnostic algorithms. In section 6.8, precision and recall statistical measures will be applied to evaluate the accuracy of the results.

6.7.1 Enhancement of Diagnostic Algorithms

While applying the diagnostic algorithms, discussed in Chapter 3, there was an opportunity for enhancements. Overlaps in identifying the evicted tasks have been noticed in *Algorithm 2b: Cause 2 Eviction Identifier* and *Algorithm 3b: Cause 3 Eviction Identifier*. Thus, the attempt to minimize the overlaps is described below. This could explain the gap found in the overall results in Chapter 4.

Table 6.9 Output of the Auditor

	Scenario 1		**Scenario 2**			**Scenario 3**		
	Total Tasks Found	***Evicted Tasks Found***	***Total Tasks Found***	***Evicted Tasks Found***		***Total Tasks Found***	***Evicted Tasks Found***	
Run 1	4208	259	3735	C1	C2	3131	C1	C3
				900	13		307	58
Run 2	4530	266	4076	C1	C2	2341	C1	C3
				960	15		91	60
Run 3	4501	421	4328	C1	C2	2687	C1	C3
				1048	20		75	55
Run 4	4653	297	4035	C1	C2	3077	C1	C3
				1012	22		182	36
Run 5	4538	319	4049	C1	C2	2596	C1	C3
				997	23		159	55

When identifying the evicted tasks because of both Cause 2 (Increase in Resource Request) or Cause 3 (Demand Exceeds the Physical Capacities), all tasks evicted due to Cause 1 (Take Over by Higher Priority Tasks) should be excluded before applying the algorithms 2b and 3b, shown in tables 6.10 and 6.12, respectively.

Further, because the simulation tool has been configured to apply the threshold usage level, Cause 3 diagnostic algorithms, described in Chapter 3 and implemented in Chapter 4, have been enhanced to be applicable for the normal environment setup, shown in tables 6.11 and 6.12.

Table 6.10 Enhanced Algorithm 2b: Cause 2 Eviction Identifier.

```
1. FOR each task in TaskEvent table (TEv) with an increase to their
   resources' request, until end of period
2.      IF ((Status = evict)
        AND (Task_timestamp (updated) < Task_timestamp (evited) <=
        (Task_timestamp (updated) + next time interval))
        AND Task priority (updated) > Task priority (evicted))
        AND Task ID NOT IN Cause1EvictedTasks table
3.          THEN display Task ID, Task_timestamp
4.      END IF
5. END FOR
```

Table 6.11 Enhanced Algorithm 3a: Cause 3 Capacities Calculator**.**

Comparing the total physical capacities with the resources usage. Once the usage reaches threshold (80%), store physical machine ID with timestamp of overload in Overloaded Table (OT).

```
1. FOR each physical machine (PM) in Sc3dataset table, until end of period
2.      Find total CPU/RAM usage in every interval
3.      IF CPU/RAM usage >  threshold level
4.          Store PM ID, timestamp in Overloaded table
5.      END IF
6. END FOR
```

Table 6.12 Enhanced Algorithm 3b: Cause 3 Eviction Identifier.

Per every overloaded physical machine in overload table OT, find all evicted tasks within one interval of overload in the same machine.

```
1. FOR each physical machine in OT, until end of period
2.      FOR each task in Sc3dataset table (ST) hosted in an overloaded a
        physical machine (PM) that is in OT, until end of period
3.          IF (ST.Status = evict)
            AND (PM_timestamp < Task_timestamp  <= (PM_timestamp +
            next time interval))
            AND Task ID NOT IN Cause1EvictedTasks table
4.              THEN display ST.Task_timestamp, ST.Task ID
5.          END IF
6.      END FOR
7. END FOR
```

6.8 Precision and Recall Statistical Measures

The simulation facilitated the generation of 15 Cloud test datasets that captured specific behaviours for task eviction. The developed diagnostic algorithms make use of PROV-TE. This has proved to be helpful by both the ability of auditing the datasets and identifying evicted tasks and links to possible causes.

In order to evaluate the accuracy of the diagnostic algorithms, precision and recall statistical measures have been applied. Precision is a measure of the reliability of the diagnostic algorithms to only identify the relevant evicted tasks for each cause. Recall is a measure of the sensitivity of the diagnostic algorithms to retrieve and identify the highest possible number of relevant evicted tasks for a specific cause.

$$\text{Precision} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)+False Positive(FP)}} \times 100 \qquad (6.1)$$

$$\text{Recall} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)+False Negetive (FN)}} \times 100 \qquad (6.2)$$

TP is the number of relevant evicted tasks stated in the simulation log and captured by the Auditor. FP is the number of irrelevant evicted tasks stated by the simulation log but captured by the Auditor. FN is the number of relevant evicted tasks stated in the simulation log but NOT captured by the Auditor. TN is the number of irrelevant evicted tasks stated by the simulation log and NOT captured by the Auditor.

TP, FP, and TN were calculated by comparing the output of the simulation, Table 6.6, with the output of the Auditor, Table 5.9. For example, looking at these two tables, in Run 1 of Scenario 2, the Auditor as able to identify all tasks, 3735, and

also classify the evicted tasks based on the specific causes, C1 and C2. For C1, 900 (TP) evicted tasks out of 904 have been identified (FN = 4).This gives 100% for precision and 99% for recall. For C2, 13 evicted tasks out of 20 were identified which makes precision 100% and recall 65%.

The precision and recall of C1 related algorithms were calculated in every scenario because all datasets captured C1 task eviction behaviour whereas C2 task eviction behaviour was captured in only Scenario 2 and C3 task eviction behaviour was captured in only Scenario 3.

6.8.1 Scenario 1 Analysis

Table 6.13 shows the simulated datasets of scenario 1. There is only one cause and the algorithms have identified all evicted tasks due to this cause.

Table 6.13 Scenario 1 Mean Precision and Recall.

C1 Algorithms	***Relevant Tasks (Simulated)***	***Irrelevant Tasks (Simulated)***
Relevant Tasks (Auditor)	TP = 312.4 STD DEV = ± 65	FP = 0 STD DEV = ± 0
Irrelevant Tasks (Auditor)	FN = 0 STD DEV = ± 0	TN = 0 STD DEV = ± 0
Precision	100%	
Recall	100%	

In Table 6.13, the mean TP of all 5 runs is 312.4 identified evicted tasks with a standard deviation of ±65 tasks. The mean FP, FN, and TN is 0 and so is the standard deviation. Applying Equations 5.1 and 5.2, the precision and recall are both 100%.

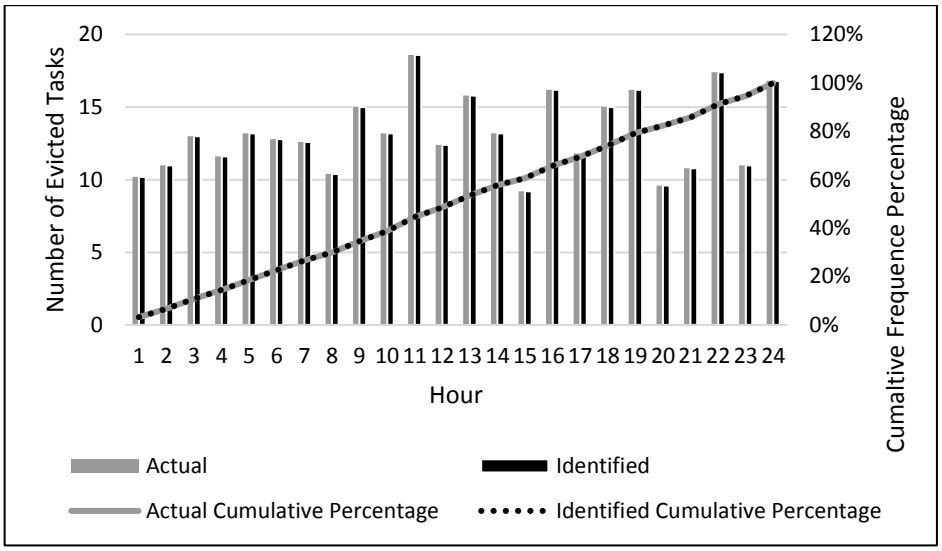

Figure 6.7 Cumulative average task evictions over all 5 runs of Scenario 1 Cause 1

In Figure 6.7, the average actual and identified evicted tasks for the whole simulation period (24 hours) on an hourly basis can be observed for Scenario 1 Cause 1. Also, it can be seen that 100% of evicted tasks were identified. It shows the cumulative average of actual and identified evicted tasks of all 5 runs of Scenario 1, Cause 1. The bars represent the average number of evicted tasks per hour. Grey bars represent the actual number of evicted tasks. Black bars

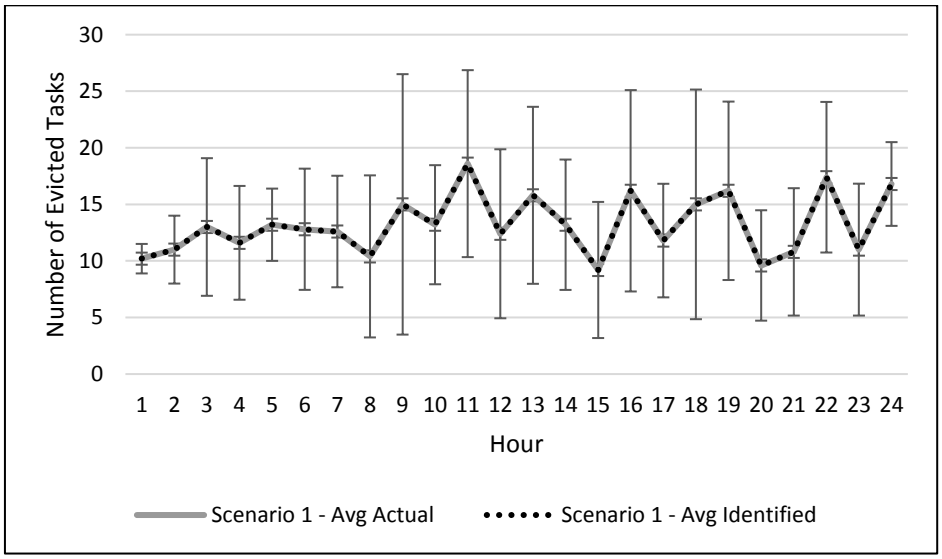

Figure 6.8 Average of actual and identified evicted tasks per hour, showing the variance across all 5 runs of Scenario 1 Cause 1

represent the found number of evicted tasks. There are two lines, which overlap each other in this case because 100% of evicted tasks were identified. The same colouring representation applies for the rest of the figures, grey for actual and black for found evicted tasks. The lines are cumulative frequency representation to show the percentage of match between the actual and the found.

In Figure 6.8, the randomization of the generated data can be seen, around 10-20 tasks per hour on average. It shows the actual (grey line) and identified (dotted black line) average number of evicted task with standard deviation across all 5 runs for this scenario for Cause 1. They overlap because 100% of evicted tasks were identified.

6.8.2 Scenario 2 Analysis

Table 6.14 Scenario 2 Cause 1 Mean Precision and Recall.

C1 Algorithms	***Relevant Tasks (Simulated)***	***Irrelevant Tasks (Simulated)***
Relevant Tasks (Auditor)	TP = 981.4 STD DEV = ± 54	FP = 2 STD DEV = ± 3
Irrelevant Tasks (Auditor)	FN = 2.2 STD DEV = ± 3.1	TN = 0 STD DEV = ± 0
Precision	99%	
Recall	99%	

Table 6.15 Scenario 2 Cause 2 Mean Precision and Recall.

C2 Algorithms	***Relevant Tasks (Simulated)***	***Irrelevant Tasks (Simulated)***
Relevant Tasks (Auditor)	TP = 18.6 STD DEV = ± 4.3	FP = 0 STD DEV = ± 0
Irrelevant Tasks (Auditor)	FN = 14.4 STD DEV = ± 8.4	TN = 0 STD DEV = ± 0
Precision	100%	
Recall	56%	

Table 6.14 summarizes the mean precision and recall of scenario 2 Cause 1 across all runs. It can be seen that the diagnostic algorithms of C1 are quite promising with 99% in both precision and recall. In Table 6.14, the mean TP of all 5 runs is 981.4 identified evicted tasks with a standard deviation of ±54 tasks. The mean FP of all 5 runs is 2 identified evicted tasks with a standard deviation of ±3 tasks. The mean FN of all 5 runs is 2.2 identified evicted tasks with a standard deviation of ±3.1 tasks. The mean TN is 0 and so is the standard deviation. In Table 6.15, the mean TP of all 5 runs is 18.6 identified evicted tasks with a standard deviation of ±4.3 tasks. The mean FN of all 5 runs is 14.4 identified evicted tasks with a standard deviation of ±8.4 tasks. The mean TN and FP is 0 and so is the standard deviation. In Table 6.15, C2 diagnostic algorithms have returned precisely the relevant evicted tasks but failed to pick up 44% of the evicted tasks linked to C2.

It can be seen from Figure 6.9 that 99% of evicted tasks were identified. It shows the cumulative average of actual and identified evicted tasks of all 5 runs of Scenario 2, Cause 1. The bars represent the hourly average number of evicted

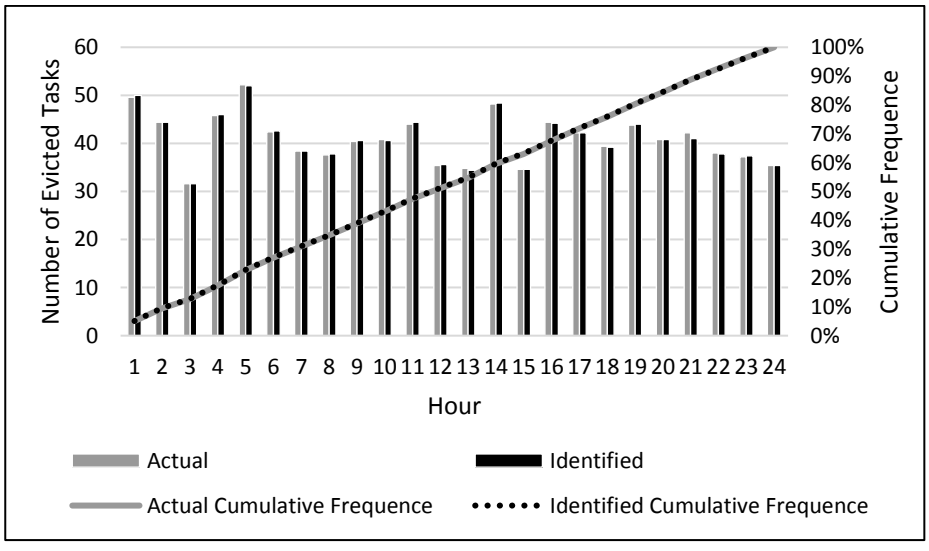

Figure 6.9 Cumulative average task evictions over all 5 runs of Scenario 2 Cause 1

tasks across all 5 runs. There are two lines, which overlap each other in this case. The lines are cumulative frequency representation to show the percentage of the match between actual and found number of evicted tasks.

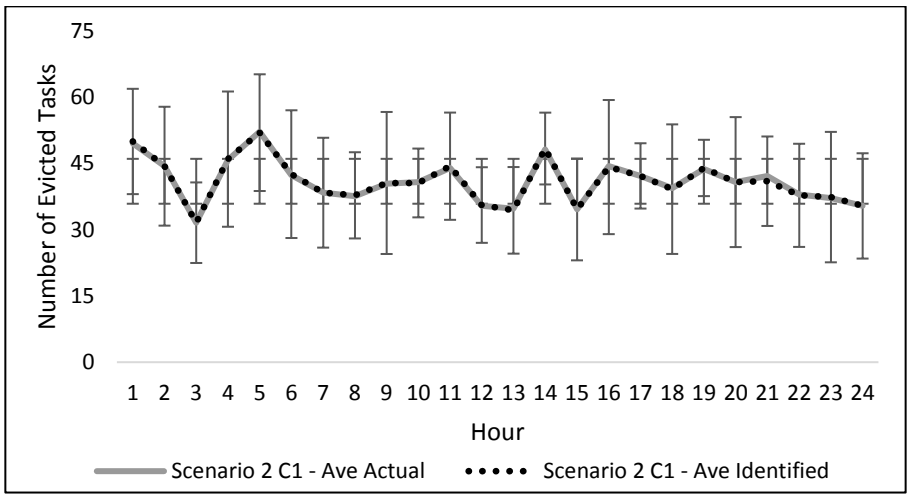

Figure 6.10 Average of actual and identified evicted tasks per hour, showing the variance across all 5 runs of Scenario 2 Cause 1

Figure 6.10 shows the randomization of the generated data, 32-47 tasks per hour on average. The average actual and identified evicted tasks for the whole simulation period (24 hours) on an hourly basis of Cause 1 are shown in the figure. The dotted black line represent the average number of identified evicted tasks per hour. The grey line represent the actual number of evicted tasks per hour. They overlap because 99% of evicted tasks were identified.

It can be seen from Figure 6.11 that 56% of evicted tasks were identified. The figure shows the cumulative average of actual and identified evicted tasks of all 5 runs of Scenario 2, Cause 2. The bars represent the hourly average number of tasks across all 5 runs. The lines are cumulative frequency representation to show the percentage of the match between actual and found number of evicted tasks.

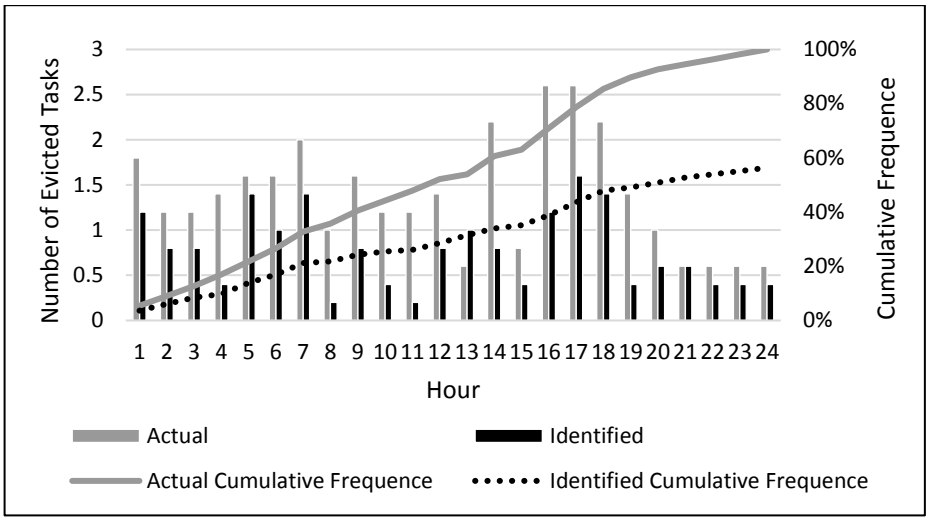

Figure 6.11 Cumulative average task evictions over all 5 runs of Scenario 2 Cause 2

In Figure 6.12, the average number of the identified evicted tasks (56% presented in Table 6.15) that are linked to Cause 2 can be observed on an hourly basis. The number of evicted tasks that have not been picked up by the framework can be observed in an hourly basis. The average number of evicted tasks per hour is based on the devolved randomizer and the task eviction behaviour models.

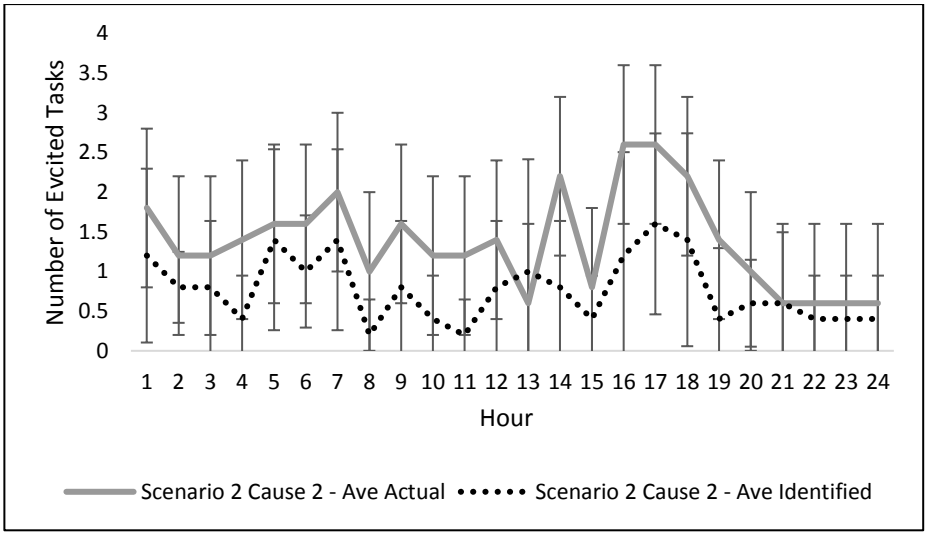

Figure 6.12 Average of actual and identified evicted tasks per hour, showing the variance across all 5 runs of Scenario 2 Cause 2

6.8.3 Scenario 3 Analysis

In Table 6.16, C1 diagnostic algorithms of Scenario 3 are able to identify relevant evicted tasks. The recall percentage of C3 diagnostic algorithms is high, 98%, which means it is capable of identifying the relevant evicted tasks as shown in Table 6.17. However, its precision measure is 40%, as seen in Figures 6.15 and 6.16. In Table 6.16, the mean TP of all 5 runs is 119.4 identified evicted tasks with a standard deviation of ±126.2 tasks. The mean FP of all 5 runs is 43.4 identified evicted tasks with a standard deviation of ±37.6 tasks. The mean FN of all 5 runs is 3.8 identified evicted tasks with a standard deviation of ±8.4 tasks. The mean TN is 0 and so is the standard deviation. In Table 6.17, the mean TP

Table 6.16 Scenario 3 Cause 1 Mean Precision and Recall.

C1 Algorithms	***Relevant Tasks (Simulated)***	***Irrelevant Tasks (Simulated)***
Relevant Tasks (Auditor)	TP = 119.4 STD DEV = ± 126.2	FP = 43.4 STD DEV = ± 37.6
Irrelevant Tasks (Auditor)	FN = 3.8 STD DEV = ± 8.4	TN = 0 STD DEV = ± 0
Precision	73%	
Recall	97%	

Table 6.17 Scenario 3 Cause 3 Mean Precision and Recall

C3 Algorithms	***Relevant Tasks (Simulated)***	***Irrelevant Tasks (Simulated)***
Relevant Tasks (Auditor)	TP = 52 STD DEV = ± 9.2	FP = 78 STD DEV = ± 55.5
Irrelevant Tasks (Auditor)	FN = 0.8 STD DEV = ± 1.7	TN = 0 STD DEV = ± 0
Precision	40%	
Recall	98%	

of all 5 runs is 52 identified evicted tasks with a standard deviation of ±9.2 tasks. The mean FP of all 5 runs is 78 identified evicted tasks with a standard deviation of ±55.5 tasks. The mean FN of all 5 runs is 0.8 identified tasks with a standard deviation of ±1.7 tasks. The mean TN is 0 and so its standard deviation.

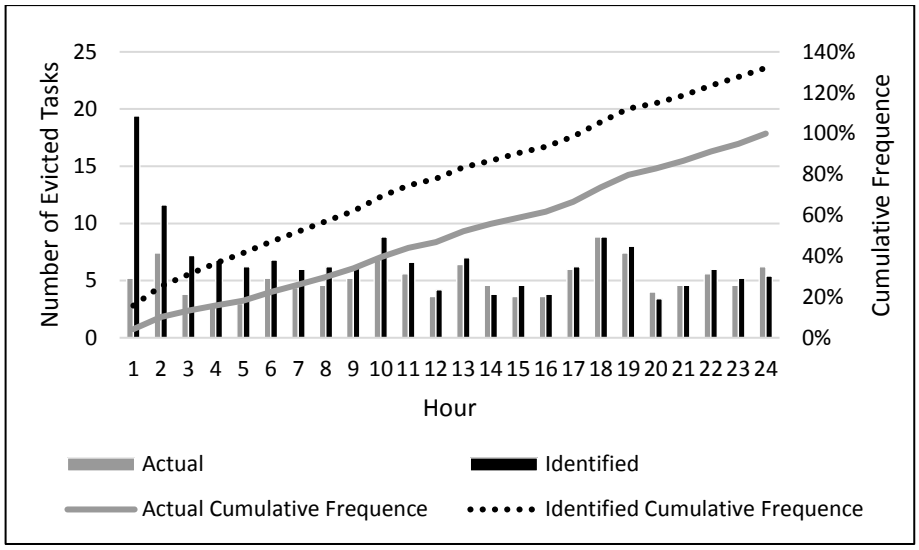

Figure 6.13 Cumulative average task evictions over all runs of Scenario 3 Cause 1

In Figure 6.13, the average actual and identified evicted tasks for the whole simulation period (24 hours) for Scenario 3 Cause 1 on an hourly basis can be observed. It can be seen that the diagnostic algorithms for C1 have identified more evicted tasks than expected with a 73% precision. It can also be seen from the two lines that about 40% of irrelevant evicted tasks were identified. The lines are cumulative frequency representation to show the percentage of the match between actual and found number of evicted tasks.

In Figure 6.14, the randomization of the generated data can also be seen. The figure shows the hourly average number of the actual and identified task eviction with standard deviation across the 5 runs of Scenario 3 Cause 1. It can be observed that in the first 11 hours the precision of the algorithms was not high

unlike the remaining hours. However, as shown in both figures 6.13 and 6.14, there is probably an overlap in terms of the identified causes as 40% of which are irrelevant.

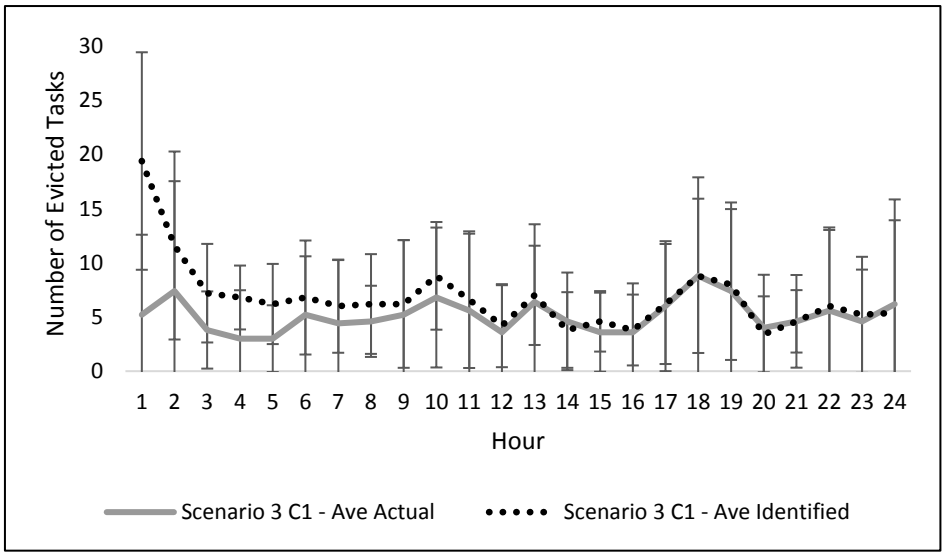

Figure 6.14 Average of actual and identified evicted tasks per hour, showing the variance across all runs of Scenario 3 Cause 1

In Figure 6.15, the average actual and identified evicted tasks for the whole simulation period (24 hours) for Scenario 3 Cause 3 on an hourly basis can be observed. Looking at the dotted black (identified) and solid grey (actual) lines,

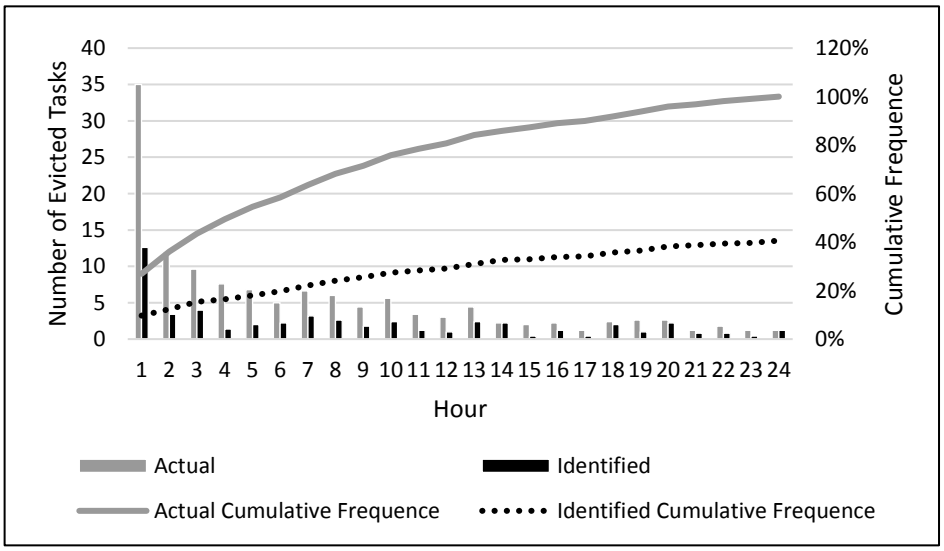

Figure 6.15 Cumulative average task evictions over all runs of Scenario 3 Cause 3

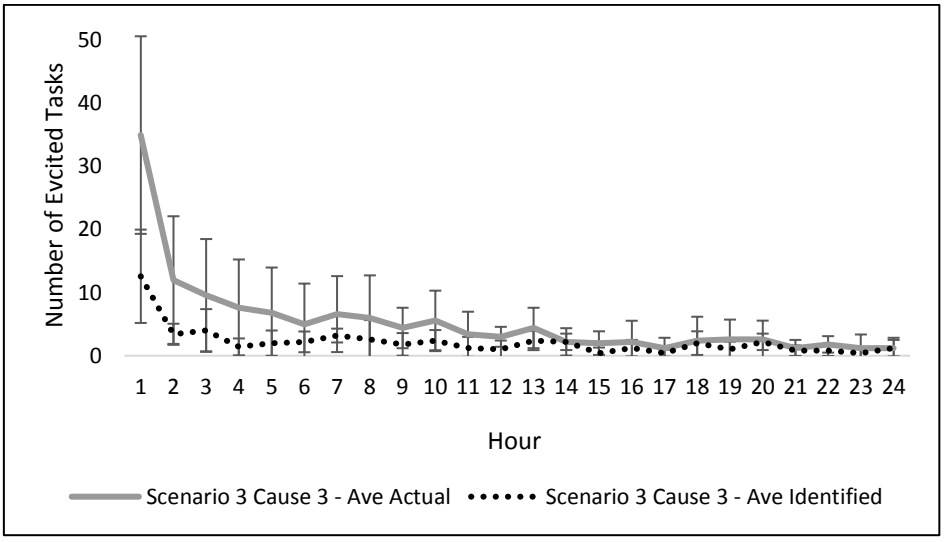

Figure 6.16 Average of actual and identified evicted tasks per hour, showing the variance across all 5 runs of Scenario 3 Cause 3

only 40% of the evicted tasks were identified. This is a limitation of the diagnostic algorithms and is discussed later.

In Figure 6.16, the randomization of the generated data can also be seen. The figure shows the hourly average number of the actual and identified task eviction with standard deviation across the 5 runs of Scenario 3 Cause 3. It can be observed that the precision increases overtime. However, almost 60% of relevant evicted tasks linked to Cause 3 have not been picked up by the diagnostic algorithms, refer to Table 6.17.

6.9 Overall Analysis

For every simulation run of every scenario, the Auditor can generate files for each cause which include the IDs of the evicted tasks and their physical and virtual host IDs which can be further investigated. Also, the Auditor can order the causes in terms of extent of impact on the system. From Tables 6.6 and 6.9, it can be observed the most dominant cause is C1 which is the Arrival of Higher

Priority Tasks and the least dominant cause is C2 which is Increase in Resource Request.

The precision and recall of C1 diagnostic algorithms are relatively high across all scenarios, as seen in Tables 6.13, 6.14, and 6.16. In Table 6.15, the 56% recall of C2 diagnostic algorithms could be because the algorithms' focus is looking at one cause instead of looking at the two causes at the same time. This may have led the algorithms to consider a number of relevant tasks as irrelevant. In Tables 6.16 and 6.17, the 73% and 40% precision percentages means that irrelevant evicted tasks were considered as relevant by the Auditor.

The cumulative frequency distribution in the above figures has been calculated by adding each frequency from a frequency distribution table to the sum of its predecessors. Each graph has its own frequency distribution table.

6.10 Possible Direction for Better Accuracy

In order to further enhance the accuracy of the diagnostic algorithms, the investigatory results of each cause for each Scenario have been combined which has formulated an assumption.

Figure 6.17 is a combination of the Figures 6.9 and 6.11. It shows that almost 100% of all evicted tasks due to C1 and C2 could be identified. The figure shows Cumulative average task evictions over all simulations of Scenario 2, combining Cause 1 and Cause 2.

Figure 6.18 is a combination of Figures 6.13 and 6.15. It also shows that almost 90% of all evicted tasks due to C1 and C3 could be identified. The figure shows

Cumulative average task evictions over all simulations of Scenario 3, combining Cause 1 and Cause 3.

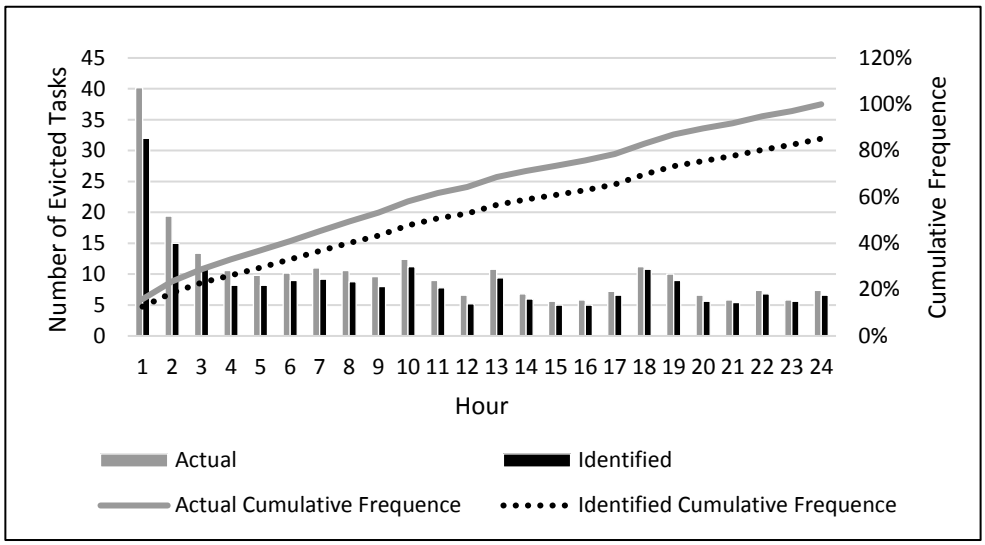

Figure 6.17 Cumulative average task evictions over all simulations of Scenario 3, combining C1 and C3

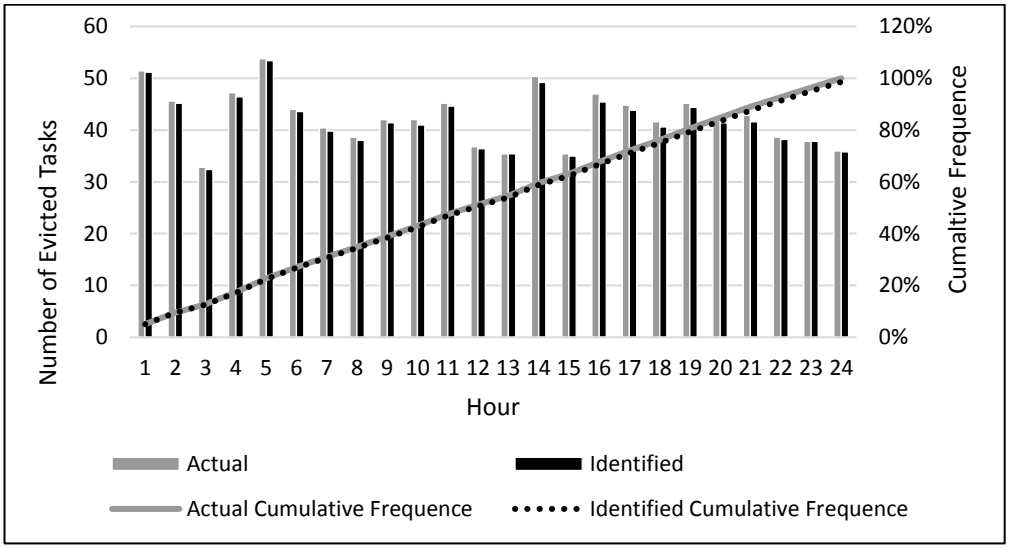

Figure 6.18 Cumulative average task evictions over all simulations of Scenario 2, combining C1 and C2

This suggests an assumption that running a hybrid algorithm that investigates two or more causes could return better results with higher precision and recall instead of auditing each cause separately. When identifying the evicted tasks and the linked cause(s), the algorithm could dynamically discard irrelevant

evicted tasks and be able to distinguish the relevant linked cause which could potentially minimize the noise in the results. Further investigation is needed to evaluate and test this assumption.

6.11 Possible Methods for Evaluation

There are other methods that can be used to evaluate the diagnostic algorithms other than simulation, such as using another real dataset, a subset of the same dataset used for exploration and learning, or a Cloud testbed to generate usage data. First, real Cloud usage datasets are a very excellent option but researchers may face a risk of making assumptions due to data obfuscation. For this research, there was no other publicly available Cloud usage dataset. Second, dividing a dataset into two sets, one for experimenting and the other for evaluation, is also a good method. However, the characteristics, the features, and the variables are the same across the two subsets which makes the scope very specific and the evaluation not thorough. This method was not possible because the full 29-day dataset has been used for learning. However, an extended version of the same dataset can be used if available. Third, Cloud testbeds, local or global, are widely used as testing platforms to experiment with new services leveraging flexible compute node and network provisions capabilities. The testbed itself is not a method for evaluation but the generated usage dataset is which makes it one example of the first method, real Cloud usage dataset. The challenges of this method are the complications of administrative rights and the suitability and volume of the generated usage datasets.

6.12 Summary

This Chapter has presented the evaluation step of the proposed diagnostic algorithms. The framework's aim is to find the causes of Task Eviction in a data centre. The contribution of this Chapter is the evaluation of the diagnostic algorithms using a simulation tool, SEED. The simulation tool has been used to generate 15 different Cloud test datasets with different task eviction behaviours. The Framework, PROV-TE and Diagnostic Algorithms, have been applied on these datasets and the found results have been compared with the simulation results. Finally, the results have been statistically analysed using precision and recall measures to find the levels of sensitivity and reliability. The average precision and recall of the diagnostic algorithms are 83% and 90%, respectively. Although the diagnostic algorithms are able to identify the causes of task eviction fairly precisely, there are still limitations relating to the overlapping of identified causes for evicted tasks. It ended with giving a brief summary of the possible approaches that could have been used for the evaluation other than simulation.

Chapter 7

Conclusion and Future Work

The first section of this Chapter presents a summary of the previous chapters that illustrate the utilization of provenance in Cloud computing which led to proposing the novel Provenance-Driven Diagnostic Framework as an approach to audit and identify the causes of Task Eviction in Cloud computing. The second section of this Chapter presents the research contributions of this study. The third section presents the summary of achievements of this research by revisiting the research questions. Finally, this Chapter concludes by discussing the research limitations and possible future work to further extend this research.

7.1 Summary of Chapters

Chapters 2 and 3 have provided a detailed summary about Cloud computing. They presented an overview on Cloud computing deployment modes and delivery models and the challenges in adopting the Clouds. They also discussed in detail the virtualization feature which is an essential key component in Cloud computing. Besides, the benefits of virtualization to both the consumer and providers has been presented, along with the method of utilizing resources which is over-commitment. The issue of Physical Machine Overloads has been discussed in addition to the overload mitigating strategies; namely Resource Stealing, Quiescing, Live Migration, Streaming Disks, and Network Memory, and Task Eviction.

The concept of provenance and the two standards of provenance models have been discussed, OPM and W3C RROV. The potential advantages of provenance to Cloud computing have been presented, along with the challenges and issues

in adopting provenance in the Clouds. Next, a number of research projects that utilized provenance in the Clouds have been presented. Lastly, the Chapter provided a literature review of the uses of provenance and the W3C PROV model in the Clouds. The review of the literature revealed that PROV model is adoptable in the Clouds but it has never been utilized with regards to the issue of Task Eviction which led to the proposing of the Provenance-Driven Diagnostic Framework.

Chapter 4 proposed the novel Provenance-Driven Diagnostic Framework and the three-phased methodology to construct it. The underpinning philosophy of the framework has been explained. The real Cloud dataset used for learning has been empirically analysed in detail and presented. The data pre-processing of the framework and working scenarios of tracing of task eviction workflow for the five Task Eviction causes as an example of demonstrating PROV-TE model have been discussed. Further, 10 diagnostic algorithms which are used to identify the evicted tasks and the linked causes have been discussed.

Chapter 5 has presented an exploratory experiment which illustrates the use of the Auditor for acquiring a deeper understanding of the causes of Task Eviction. It started by giving a deeper analysis of the used Cloud dataset for the application of the diagnostic algorithms. It then described the aim and hypothesis of the experiment. Then it presented the application of the diagnostic algorithms in five separate investigations, each focusing on one Task Eviction cause. It showed the results obtained from the application of the diagnostic algorithms which quantifies the extent of every cause in the dataset. It also explained how PROV-TE contributes specifically to every cause and guides the investigations. It

concluded with identifying the most dominant and least dominant causes for Task Eviction observed in Google's dataset.

One of the challenges in this chapter is the volume of the dataset which led to scaling down the data to be queried. This has increased the performance (query response time) and was deemed small enough to fulfil the objections of the investigations. Provenance in general faces a challenge with level of granularity, as discussed in section 3.3.1. In section 7.4, a number of possible directions and techniques has been introduced as future work which could help with recording and using provenance data large datasets, such as graph database and NoSQL.

This chapter has studied and analysed a dataset that applies Task Eviction mitigating strategy. As a result, PROV model has been extended to PROV-TE. In case another dataset applies another mitigating strategy, such as Live Migration, the same generic framework, section 4.4, could be used and the steps could be followed to extend PROV model to PROV-LM and so on, as explained in section 4.9. One of the future directions, section 7.4, the framework could be extended to include all six mitigating strategies. The challenge that could be faced is the selection of relevant attributes. This could increase the number of iterations and tests until the expected accuracy level is reached.

Chapter 6 has presented the evaluation step of the proposed diagnostic algorithms. The framework's aim is to find the causes of Task Eviction in a data centre. The contribution of this Chapter is the evaluation of the diagnostic algorithms using a simulation tool, SEED. The simulation tool has been used to generate 15 different Cloud test datasets with different task eviction behaviours. The Framework, PROV-TE and Diagnostic Algorithms, have been applied on

these datasets and the found results have been compared with the simulation results. Finally, the results have been statistically analysed using precision and recall measures to find the levels of sensitivity and reliability. The average precision and recall of the diagnostic algorithms are 83% and 90%, respectively. Although the diagnostic algorithms are able to identify the causes of task eviction fairly precisely, there are still limitations relating to the overlapping of identified causes for evicted tasks.

The scale of the simulation can be generalized to larger environments with more PMs and VMs, generating huge volumes of data. This aims to demonstrate the feasibility of massive-scale simulation for implementing provenance-based techniques.

7.2 Research Contributions

The major contributions of the research presented in this thesis are summarized as follows:

- *A provenance framework that acts as a diagnostic tool to find the causes of an overload in the Clouds by two steps. First, the PROV model was extended to represent a task eviction mitigating strategy, refer to section 4.6.1. Second, relevant attributes to the strategy were identified, refer to section 4.6.* Over-commitment of resources is a beneficial policy for Cloud providers because it maximizes profits and utilizes the idle resources. Overload is an inevitable consequence of over-commitment if it was not administered carefully. The way the mitigating strategies are used to cope with overload has room for improvements, refer to section 3.2. One of which is to act proactively and identify the causes of overload by empirically studying the strategies backwards to

quantify the impact. This thesis has presented a novel framework and is provenance-driven which was tested and evaluated. This framework is capable in identifying the causes for task and job evictions. Knowing the causes and their behaviour could help mitigate them in the first place which proactively minimizes the number of overload instances.

- *A computational version of the model for reasoning. A querying platform and algorithms were developed to find the causal relationship between causes and tasks by identifying the evicted tasks and the linked causes.* These are the second and third parts of the proposed framework. The extended PROV model (PROV-TE) underpins a number of diagnostic algorithms which were developed by using a real Cloud dataset for learning. The diagnostic algorithms have been operationalized using SQL queries. With the help of both PROV-TE and the diagnostic algorithms, it has been proven possible for evicted tasks and the linked causes to be identified. The potential user for this framework is a Cloud provider. The framework helps providers to efficiently manage their data centres, specifically with regards to Task Eviction policy. The Auditor presented in this research, see section 4.9, can be utilized to be applicable for another mitigating strategy following the systematic steps presented in Chapter 3.

- *The modelling of Task Eviction behaviors in a Cloud datacentre with provenance data for a simulations*. A controlled Cloud environment was built using the simulation tool, SEED. The configurations of both Amazon EC2 Cloud and Google Cloud have been combined to form a hypothetical configuration and used as the basis for the built environment. Three Task Eviction behaviours (causes) have been modelled into the simulation tool. 15 heterogeneous and randomized Cloud datasets were generated and used for the evaluation of the

diagnostic algorithms. The transferability of PROV-TE and the diagnostic algorithms has been demonstrated by being applied on different datasets using simulation, see Chapters 5 and 6.

7.3 Summary of Achievements

The three research questions of this thesis were discussed in Chapter 1, each question and the success of this research answering it is listed below:

Q1. How to formulate a suitable diagnostic provenance model that will help check the causes of overload in a Cloud platform?

In this thesis, two standard models for provenance have been reviewed and analysed, refer to section 2.3.2. PROV model has been chosen as the abstract provenance model for the envisioned framework because it is a W3C standard. Related state-of-the-art work that used PROV in Cloud computing has been critically reviewed to understand how PROV can be extended. Also, the largely publicly available real Cloud datasets has been empirically analysed and used for learning. As a result, PROV has been extended to be applicable on Cloud infrastructures and named PROV-TE model, please see section 4.6. PROV-TE is one of three parts of the Provenance-Driven Diagnostic Framework.

Q2. How to operationalize the model?

PROV-TE is a theoretical model. In order to test its potential, it needs to be operationalized. By empirically studying Google's 29-day Cloud usage dataset, 10 diagnostic algorithms have been developed taking PROV-TE model as the underpinning basis, refer to sections 4.3 and 4.8. These algorithms have been applied on the dataset to test whether PROV-TE's reasoning power helps quantify the extent of Task Eviction causes, see section 4.4. Each algorithm has

been translated in to an SQL query and SQLite3 has been used as the query platform. The Auditor has been presented in Chapter 4 is an instantiation of the framework and it demonstrates how the frameworks fits in a data centre.

Q3. How to evaluate the proposed framework?

It is not effective to use the same dataset for evaluating the framework. Due to the lack of publicly available Cloud dataset, simulation has been used as a method for the evaluation, see section 6.2. A simulation tool has been chosen and used to randomly generate 15 different Cloud datasets, refer to sections 6.3 – 6.6. These datasets have been used to test the accuracy and reliability of the proposed Provenance-Driven Diagnostic Framework, see section 6.8. In section 6.11, the possible approaches that could have been used for the evaluation other than simulation have been presented.

The outcomes of this framework could be used by Cloud providers to make better informed decisions and to identify the maximum over-commitment ratio that fits the data center infrastructure. Providers could use the framework to test the infrastructure until they reach the most suitable and less damaging OCR. Cloud brokers could make use of the framework to choose the suitable providers for the clients. Cloud auditors can use the framework as an auditing approach that systematically extends a standardized provenance model to make sense of historical data. Lastly, Cloud consumers (users) can use the framework to audit the performance and availability of the Cloud services. Access to the IaaS Infrastructure Monitoring component is required.

Table 7.1 below revisits Table 3.1 and shows the comparison between the different possible methods for causes identification and the proposed provenance-driven diagnostic framework, named PROV-TE framework.

Table 7.1 Comparison of Different Possible Methods for Overload Causes Identification and PROV-TE Framework

Method (page in this thesis)	Capability						Scope		
	Prediction	Detection	Diagnosis	Mitigation	Prevention	Healing	Software	Hardware	Network
FTM by Jhawar et al [105], [106] / (p. 43) non-provenance based	No	Yes	No	No	Yes	Yes	Yes	Yes	Yes
Dai et al [107] / (p. 44) non-provenance based	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No
TCloud by Verissimo et al FTCloud by Zheng et al [97], [108] / (p. 44) non-provenance based	No	Yes	No	No	Yes	No	Yes	Yes	No
Chopstix by Bhatia et al [112] / (p. 46) non-provenance based	No	Yes	Yes	No	No	No	Yes	Yes	No
Fay by Erlingsson et al [113] / (p. 46) non-provenance based	No	No	Yes	No	No	No	Yes	Yes	No
D3S by Liu et al [115] / (p. 46) non-provenance based	Yes	Yes	Yes	No	No	No	Yes	No	No
Pip by Reynolds et al [116] / (p. 47) non-provenance based	No	Yes	Yes	No	No	No	Yes	No	No
CPN by Li et al [129] / (p. 52) Provenance based	No	No	Yes	No	No	No	Yes	No	Yes
DTaP by Zhou et al [130] / (p. 52) Provenance based	No	No	Yes	No	No	No	No	No	Yes
S2Logger by Suen et al [133] / (p. 53) Provenance based	No	Yes	Yes	No	No	No	Yes	Yes	Yes
PROV-TE framework [138], [161] / (Chapter 4) Provenance based	No	Yes	Yes	No	No	No	Yes	Yes	No

7.4 Limitations and Future Work

The work in this thesis is a beneficial first step in measuring the extent of the causes of overload for all six mitigating strategies. Although the proposed framework is able to identify the causes of task evictions fairly precisely, there are still limitations relating for example to the overlapping of identified causes for evicted tasks which can be explained by the chain of causes presented in section 5.5. PROV-TE has been developed based on Google 29-day dataset. As mentioned in Chapter 5, there are unknown number of parameters that have been deliberately obfuscated by the dataset provider. This limits the development of the framework by relying on assumptions.

By definition, simulations are approximations or abstractions of the real world [156]. Simulation models needs verification to confirm that they are correctly implemented according to the conceptual model, assumptions and specifications. Validation is also important which checks the accuracy of the simulation model and its representation and imitation of the real system [157]. For example, verification can be done by statistical testing of the final simulation output against analytical results and validation can be done by comparing simulation and real data through tests such as *t* tests. A further study is needed to verify and validate the simulation models and checks the outputs of chapter 6.

The work carried out in this thesis can be extended into several promising directions:

- The proposed Provenance-Driven Diagnostic Framework solely focuses on Task Eviction mitigating strategies. The framework could be extended to include all six mitigating strategies. The power of the framework could be

stronger as the causes of the overload could be quantified and mitigated in the first place based on extent of impact and the Cloud provider's policies.

- The working of the framework has been done manually in terms of extending the model, mapping the raw data, building the database and developing and running the diagnostic algorithms. Automating this process would make this framework dynamic. The framework could learn from provenance data on the fly and extend the model and the algorithms accordingly. For example, adding more parameters in to the database that could potentially enhance the accuracy of the diagnostic algorithms.
- As presented in Chapters 5 and 6, the diagnostic algorithms have been processed separately and each cause has been individually investigated. As seen from Figures 6.17 and 6.18 in Chapter 6, when the results of the two investigations have been combined for each scenario, the precision and recall have increased. The method of combing the diagnostic algorithms which could potentially increase levels of accuracy need to be further studied to evaluate the hypothesis.
- The proposed framework uses lightweight semantics. PROV-TE is a lightweight provenance model. Making use of heavyweight semantics by for example utilizing ontologies and RDF is an interesting future direction which could further enrich the added meaning and possible have a positive effect on auditing accuracy.
- As presented in Chapters 5 and 6, relational database and SQLite have been used for storing and querying the data. Relational databases are efficient for a data-intensive storage. For data that contains many relationships, a graph database which utilizes NoSQL could potentially be more efficient [158]. This

leads to a future work where a graph database and NoSQL could be used for storing and querying the data which could potentially enhance the reliability and sensitivity of the diagnostic algorithms.

Appendix A

First Version of PROV-TE

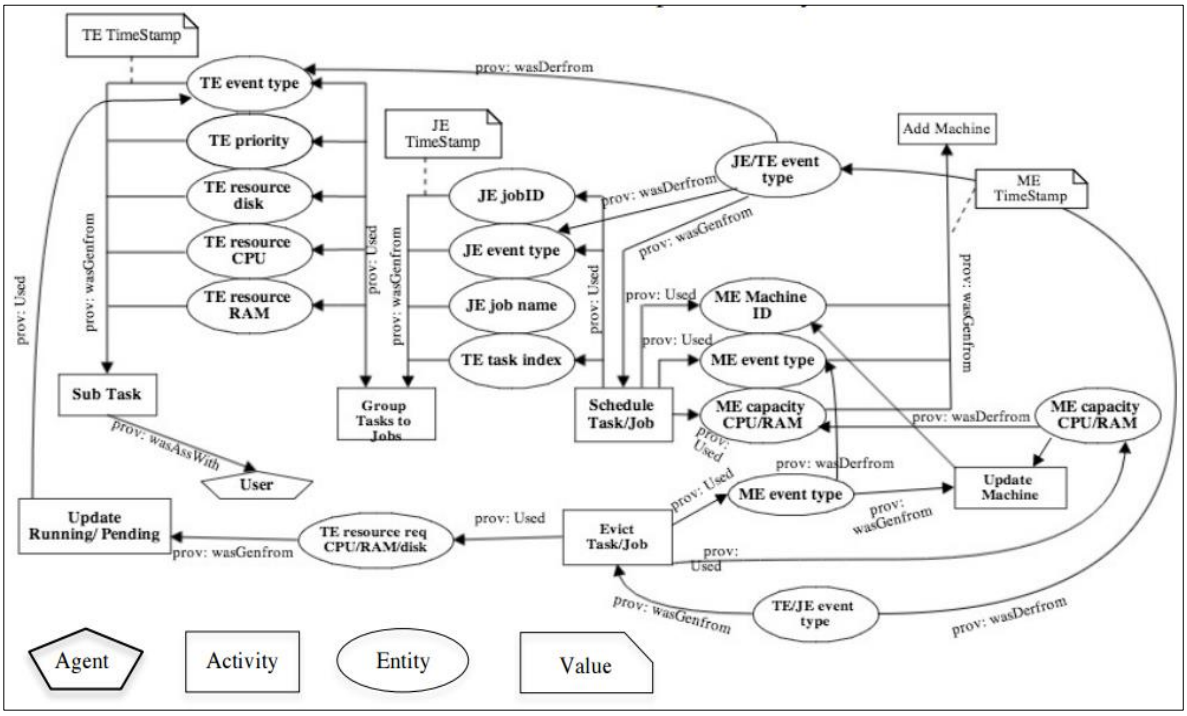

Figure A.1 First PROV-TE Model

Figure A.1 shows the first version of PROV-TE. The version contributed to the development of the diagnostic algorithms. After testing the output of the algorithms, it became clear that a number of attributes needs to be considered in the diagnosis. The results were not accurate in terms of identifying the relevant evicted tasks linked to the causes. The difference between the two versions includes corrections and more entities were included in the second version, presented in section 4.6.1. The changes are:

- Activity: Update Running/Pending has been renamed to UpdateRunningTasks.
- Agent: User is linked to Activity: UpdateRunningTasks

- Entities TE_username, TE_differentmachine, TE_schedulingclass were included in the second version.

These additions were a result of the iterations so that both PROV-TE and the diagnostics algorithms are fit-for-purpose. Also, the overall model has been redesigned for ease of understanding.

References

[1] T. Wo, Q. Sun, B. Li, and C. Hu, “Overbooking-Based Resource Allocation in Virtualized Data Center,” in *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, 2012, pp. 142–149.

[2] D. Williams, H. Jamjoom, Y.-H. Liu, and H. Weatherspoon, “Overdriver: Handling Memory Overload in an Oversubscribed Cloud,” in *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2011, p. 205.

[3] Y. Choi and H. Choi, “Evaluating the Performance of Resource Overcommitted Virtualized Systems,” in *Proceedings of the 5th International Conference on Ubiquitous Information Technologies and Applications*, 2010, pp. 1–6.

[4] S. Baset, L. Wang, and C. Tang, “Towards an Understanding of Oversubscription in Cloud,” in *USENIX Hot-ICE’12*, 2012.

[5] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, “Towards understanding heterogeneous clouds at scale: Google trace analysis,” 2012. [Online]. Available: http://www.pdl.cmu.edu/ftp/CloudComputing/ISTC-CC-TR-12-101.pdf. [Accessed: 11-Mar-2013].

[6] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, “Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation,” in *Cloud Computing*, 2nd ed., vol. 5931, M. G. Jaatun, G. Zhao, and C. Rong, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 254–265.

[7] C. Reiss, J. Wilkes, and J. Hellerstein, “Google cluster-usage traces : format + schema V2.1,” pp. 1–14, 2014.

[8] M. Andreolini, S. Casolari, M. Colajanni, and M. Messori, “Dynamic load management of virtual machines in cloud architectures,” in *Cloud Computing*, vol. 34, Springer Berlin Heidelberg, 2010, pp. 201–214.

[9] L. Moreau and P. Groth, “Provenance: An Introduction to PROV,” *Synth. Lect. Semant. Web Theory Technol.*, vol. 3, no. 4, pp. 1–129, Sep. 2013.

[10] P. Townend, D. Webster, C. C. Venters, V. Dimitrova, K. Djemame, L. Lau, J. Xu, S. Fores, V. Viduto, C. Dibsdale, N. Taylor, J. Austin, J. McAvoy, and S. Hobson, “Personalised provenance reasoning models and risk assessment in business systems: A case study,” in *IEEE 7th International Symposium on Service-Oriented System Engineering*, 2013, pp. 329–334.

[11] P. Groth and L. Moreau, “PROV-Overview: An Overview of the PROV Family of Documents,” 2013.

[12] John W. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*, 4th ed. London: SAGE Publications, 2014.

[13] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim : A Toolkit for the Modeling and Simulation of Cloud Resource Management and Application Provisioning Techniques,” *Softw. Pract. Exp.*, vol. 41, no. 1, pp. 23–50, 2011.

[14] P. Garraghan, D. McKee, X. Ouyang, D. Webster, and J. Xu, “SEED: A Scalable Approach for Cyber-Physical System Simulation,” *IEEE Trans. Serv. Comput.*, vol. 9, no. 2, pp. 199–212, Mar. 2016.

[15] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez, “Prototyping Bubba, a highly parallel database system,” *IEEE Trans. Knowl. Data Eng.*, vol. 2, no. 1, pp. 4–24, Mar. 1990.

[16] G. Manno, W. W. Smari, and L. Spalazzi, “FCFA: A semantic-based federated cloud framework architecture,” *2012 Int. Conf. High Perform. Comput. Simul.*, pp. 42–52, Jul. 2012.

[17] S. Habib, S. Hauke, S. Ries, and M. Mühlhäuser, “Trust as a facilitator in cloud computing: a survey,” *J. Cloud Comput. Adv. Syst. Appl.*, vol. 1, no. 1, p. 19, 2012.

[18] W. Venters and E. A. Whitley, “A critical review of cloud computing: researching desires and realities,” *J. Inf. Technol.*, vol. 27, no. 3, pp. 179–197, Aug. 2012.

[19] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, “A Break in the Clouds: Towards a Cloud Definition,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, p. 50, Dec. 2008.

[20] P. Mell and T. Grance, “The NIST Definition of Cloud Computing,” 2011.

[21] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “Above the Clouds : A Berkeley View of Cloud Computing,” *Dept. Electr. Eng. Comput. Sci. Univ. California, Berkeley.*, vol. 28, no. UCB/EECS-2009-28, pp. 07–013, 2009.

[22] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, “A view of cloud computing,” *Commun. ACM*, vol. 53, no. 4, p. 50, Apr. 2010.

[23] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf, “NIST Cloud Computing Reference Architecture,” 2011.

[24] M. Ahronovitz, D. Amrhein, P. Anderson, A. De, J. Armstrong, E. A. B, J. Bartlett, R. Bruklis, M. Carlson, R. Cohen, T. M. Crawford, V. Deolaliker, P. Downing, A. Easton, R. Flores, G. Fourcade, T. Hanan, V. Herrington, B. Hosseinzadeh, and S. Hughes, “Cloud Computing Use Cases. White Paper.,” *Cloud Comput. Use Case Discuss. Gr.*, no. v 4.0, pp. 1–68, 2010.

[25] C. N. Höfer and G. Karagiannis, “Cloud computing services: Taxonomy and comparison,” *J. Internet Serv. Appl.*, vol. 2, no. 2, pp. 81–94, 2011.

[26] B. P. Rimal, E. Choi, and I. Lumb, “A Taxonomy and Survey of Cloud Computing Systems,” in *2009 Fifth International Joint Conference on INC, IMS and IDC*, 2009, pp. 44–51.

[27] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu, “Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends,” in *8th IEEE International Conference on Cloud Computing, CLOUD 2015*, 2015, pp. 621–628.

[28] H. Takabi, J. Joshi, and G. Ahn, “Security and privacy challenges in cloud computing environments,” *Secur. Privacy, IEEE*, no. December, 2010.

[29] S. Pearson, “Privacy, Security and Trust in Cloud Computing,” in *Privacy and Security for Cloud Computing*, New York: Springer London, 2013, pp. 3–42.

[30] M. Henneberger and A. Luhn, “Community Clouds – supporting business ecosystems with cloud computing,” 2010.

[31] Y. Xing and Y. Zhan, “Virtualization and Cloud Computing,” in *Lecture Notes in Electrical Engineering*, vol. 143 LNEE, no. VOL. 1, Springer Berlin Heidelberg, 2012, pp. 305–312.

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

[33] D. Armstrong and K. Djemame, “Towards quality of service in the cloud,” in *Proc. of the 25th UK Performance Engineering Workshop*, 2009.

[34] J. Sahoo, S. Mohapatra, and R. Lath, “Virtualization: A survey on concepts, taxonomy and associated security issues,” in *2nd International Conference on Computer and Network Technology*, 2010, pp. 222–226.

[35] Intel IT Center, “Planning Guide: Virtualization and Cloud Computing,” 2013.

[36] H. Chen, L. Shi, J. Sun, K. Li, and L. He, “A Fast RPC System for Virtual Machines,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 7, pp. 1267–1276, Jul. 2013.

[37] D. Merkel, “Docker: lightweight Linux containers for consistent development and deployment,” *Linux J.*, vol. 2014, no. 239, p. 2, 2014.

[38] IBM Co., “Introduction to Virtualization,” 2009.

[39] E. Kafetzakis, H. Koumaras, M. A. Kourtis, and V. Koumaras, “QoE4CLOUD: A QoE-driven multidimensional framework for cloud environments,” in *2012 International Conference on Telecommunications and Multimedia, TEMU 2012*, 2012, pp. 77–82.

[40] M. Alhamad, T. Dillon, and E. Chang, “Conceptual SLA framework for cloud computing,” in *4th IEEE International Conference on Digital Ecosystems and Technologies*, 2010, pp. 606–610.

[41] Dimension Data, “Comparing Public Cloud Service Level Agreements, White Paper.,” *Dimension Data*, p. 4, 2013.

[42] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G.

Rabbani, Q. Zhang, and M. F. Zhani, “Data center network virtualization: A survey,” *IEEE Commun. Surv. Tutorials*, vol. 15, no. 2, pp. 909–928, 2013.

[43] R. Buyya, R. Buyya, C. S. Yeo, C. S. Yeo, S. Venugopal, S. Venugopal, J. Broberg, J. Broberg, I. Brandic, and I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,” *Futur. Gener. Comput. Syst.*, vol. 25, 2009.

[44] I. S. Moreno, “Characterizing and Exploiting Heterogeneity for Enhancing Energy-Efficiency of Cloud Datacenters,” University of Leeds, 2014.

[45] K. Ren, C. Wang, and Q. Wang, “Security challenges for the public cloud,” *Internet Comput. IEEE*, pp. 69–73, 2012.

[46] M. Zhou, R. Zhang, W. Xie, W. Qian, and A. Zhou, “Security and Privacy in Cloud Computing: A Survey,” in *2010 Sixth International Conference on Semantics, Knowledge and Grids*, 2010, pp. 105–112.

[47] C.-C. Feng, “Mapping Geospatial Metadata to Open Provenance Model,” *IEEE Trans. Geosci. Remote Sens.*, vol. 51, no. 11, pp. 5073–5081, Nov. 2013.

[48] J. Lacroix and O. Boucelma, “Trusting the cloud: A PROV + RBAC approach,” in *IEEE International Conference on Cloud Computing, CLOUD*, 2014, pp. 652–658.

[49] Y. L. Simmhan, B. Plale, and D. Gannon, “A survey of data provenance in e-science,” *ACM SIGMOD Rec.*, vol. 34, no. 3, p. 31, Sep. 2005.

[50] I. Foster, J. Vockler, and M. Wilde, “Chimera: a virtual data system for representing, querying, and automating data derivation,” in *Proceedings 14th International Conference on Scientific and Statistical Database Management*, 2002, pp. 37–46.

[51] J. C. Dong Yuan, Yun Yang, *Computation and Storage in the Cloud: Understanding the Trade-Offs*. USA: Elsevier Inc, 2013.

[52] O. Q. Zhang, M. Kirchberg, R. K. L. Ko, and B. S. Lee, “How to Track Your Data: The Case for Cloud Computing Provenance,” *2011 IEEE Third Int. Conf. Cloud Comput. Technol. Sci.*, pp. 446–453,

Nov. 2011.

[53] P. Buneman, S. Khanna, W. Tan, and S. Khanna, “Why and Where: A Characterization of Data Provenance,” *Lect. Notes Comput. Sci.*, pp. 316–330, 2001.

[54] P. Townend, V. Viduto, D. Webster, K. Djemame, L. Lau, V. Dimitrova, J. Xu, S. Fores, C. Dibsdale, J. Austin, J. McAvoy, and S. Hobson, “Risk Assessment and Trust in Services Computing: Applications and Experience,” in *10th IEEE International Conference on Services Computing*, 2013.

[55] I. Abbadi and J. Lyle, “Challenges for Provenance in Cloud Computing,” in *3rd USENIX Workshop on the Theory and Practice of Provenance (TaPP’11)*, 2011.

[56] M. Imran and H. Hlavacs, “Provenance in the cloud: Why and how?,” *Third Int. Conf. Cloud Comput. GRIDs, Virtualization*, pp. 106–112, 2012.

[57] W3C, “What Is Provenance - XG Provenance Wiki,” 2010. [Online]. Available: http://www.w3.org/2005/Incubator/prov/wiki/What_Is_Provenance. [Accessed: 01-Oct-2013].

[58] L. Moreau, J. Freire, J. Futrelle, R. E. Mcgrath, J. Myers, and P. Paulson, “The Open Provenance Model: An Overview,” in *Provenance and Annotation of Data and Processes*, 2008, pp. 323–326.

[59] “Provenance Challenge Wiki,” 2013. [Online]. Available: http://twiki.ipaw.info/bin/view/Challenge. [Accessed: 17-Nov-2013].

[60] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. Van den Bussche, “The Open Provenance Model core specification (v1.1),” *Future Generation Computer Systems*. Elsevier, 22-Jul-2010.

[61] T. Lebo, S. Sahoo, D. McGuinness, K. Behajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik, and J. Zhao, “PROV-O: The PROV Ontology,” 2013. [Online]. Available: http://www.w3.org/TR/prov-o/. [Accessed: 17-Oct-2013].

[62] L. Moreau, “PROV-XML: The PROV XML Schema,” 2013. [Online].

Available: http://www.w3.org/TR/2013/NOTE-prov-xml-20130430/. [Accessed: 10-Sep-2013].

[63] M. A. Sakka, B. Defude, and J. Tellez, *Document provenance in the cloud: constraints and challenges*, vol. 6164. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[64] O. Hartig and J. Zhao, “Using Web Data Provenance for Quality Assessment,” in *Proceedings of the 1st Int. Workshop on the Role of Semantic Web in Provenance Management (SWPM) at the International Semantic Web Conference (ISWC)*, 2009.

[65] Y.-W. Cheah and B. Plale, “Provenance analysis: Towards quality provenance,” in *2012 IEEE 8th International Conference on E-Science*, 2012, pp. 1–8.

[66] K. Muniswamy-Reddy, P. Macko, and M. Seltzer, “Provenance for the Cloud,” in *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, 2010.

[67] V. M. Katilu, V. N. L. Franqueira, and O. Angelopoulou, “Challenges of data provenance for cloud forensic investigations,” in *Proceedings of the 10th International Conference on Availability, Reliability and Security*, 2015, pp. 312–317.

[68] G. Birkenheuer, A. Brinkmann, and H. karl, “Risk Aware Overbooking for Commercial Grids,” Springer Berlin Heidelberg, 2010, pp. 51–76.

[69] A. Gordon, M. R. Hines, D. Da Silva, M. Ben-Yehuda, M. Silva, and G. Lizarraga, “Ginkgo: Automated, Application-Driven Memory Overcommitment for Cloud Computing.”

[70] J. O. Iglesias, L. Murphy, M. De Cauwer, D. Mehta, and B. O’Sullivan, “A Methodology for Online Consolidation of Tasks through More Accurate Resource Estimations,” in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, 2014, pp. 89–98.

[71] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, “Heterogeneity and dynamicity of clouds at scale,” in *Proceedings of the Third ACM Symposium on Cloud Computing - SoCC ’12*, 2012, pp. 1–13.

[72] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma,

"Towards autonomic workload provisioning for enterprise Grids and clouds," in *2009 10th IEEE/ACM International Conference on Grid Computing*, 2009, pp. 50–57.

[73] L. Tomas and J. Tordsson, "An Autonomic Approach to Risk-Aware Data Center Overbooking," *IEEE Trans. Cloud Comput.*, vol. 2, no. 3, pp. 292–305, Jul. 2014.

[74] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Efficient datacenter resource utilization through cloud resource overcommitment," in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2015, pp. 330–335.

[75] J. Ortigoza, F. Lopez-Pires, and B. Baran, "A Taxonomy on Dynamic Environments for Provider-Oriented Virtual Machine Placement," in *2016 IEEE International Conference on Cloud Engineering (IC2E)*, 2016, pp. 214–215.

[76] R. Ghosh and V. K. Naik, "Biting Off Safely More Than You Can Chew: Predictive Analytics for Resource Over-Commit in IaaS Cloud," in *2012 IEEE Fifth International Conference on Cloud Computing*, 2012, pp. 25–32.

[77] Y. Liu, "A Consolidation Strategy Supporting Resources Oversubscription in Cloud Computing," in *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*, 2016, pp. 154–162.

[78] D. Breitgand, Z. Dubitzky, A. Epstein, A. Glikson, and I. Shapira, "SLA-aware Resource Over-Commit in an IaaS Cloud."

[79] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource overbooking and application profiling in shared hosting platforms," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, p. 239, Dec. 2002.

[80] S. a. Baset, "Cloud SLAs: Present and Future," *ACM SIGOPS Oper. Syst. Rev.*, vol. 46, no. 2, p. 57, 2012.

[81] A. Beloglazov and R. Buyya, "Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers under Quality of Service Constraints," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 7, pp. 1366–1379, Jul. 2013.

[82] F. Caglar and A. Gokhale, “iOverbook: Intelligent Resource-Overbooking to Support Soft Real-Time Applications in the Cloud,” in *2014 IEEE 7th International Conference on Cloud Computing*, 2014, pp. 538–545.

[83] M. Unuvar, Y. N. Doganata, A. N. Tantawi, and M. Steinder, “Cloud overbooking through stochastic admission controller,” in *10th International Conference on Network and Service Management (CNSM) and Workshop*, 2014, pp. 320–323.

[84] R. Householder, S. Arnold, and R. Green, “Simulating the Effects of Cloud-Based Oversubscription on Datacenter Revenues and Performance in Single and Multi-class Service Levels,” in *IEEE 7th International Conference on Cloud Computing*, 2014, pp. 562–569.

[85] C. Colman-Meixner, C. Develder, M. Tornatore, and B. Mukherjee, “A Survey on Resiliency Techniques in Cloud Computing Infrastructures and Applications,” *IEEE Commun. Surv. Tutorials*, vol. 18, no. 3, pp. 2244–2281, Jan. 2016.

[86] L. Wang, R. A. Hosn, and C. Tang, “Remediating overload in over-subscribed computing environments,” *Proc. - 2012 IEEE 5th Int. Conf. Cloud Comput. CLOUD 2012*, pp. 860–867, 2012.

[87] L. Nogueira and L. M. Pinho, “Capacity Sharing and Stealing in Dynamic Server-based Real-Time Systems,” in *2007 IEEE International Parallel and Distributed Processing Symposium*, 2007, pp. 1–8.

[88] K. Wang, X. Zhou, K. Qiao, M. Lang, B. McClelland, and I. Raicu, “Towards Scalable Distributed Workload Manager with Monitoring-Based Weakly Consistent Resource Stealing,” in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing - HPDC ’15*, 2015, pp. 219–222.

[89] L. Wang, R. A. Hosn, and C. Tang, “Remediating Overload in Over-Subscribed Computing Environments,” in *2012 IEEE Fifth International Conference on Cloud Computing*, 2012, pp. 860–867.

[90] C. Clark, K. Fraser, S. Hand, and J. Hansen, “Live migration of virtual machines,” *Proc. 2nd …*, no. Vmm, pp. 273–286, 2005.

[91] L. Lundberg, “Performance Implications of Resource Over-Allocation During the Live Migration,” in *2016 IEEE International Conference*

on Cloud Computing Technology and Science (CloudCom), 2016, pp. 552–557.

[92] X. Wang, X. Wang, G. Xing, and C.-X. Lin, “Maximizing the detection probability of overheating server components with sensor placement based on thermal dynamics,” *Sustain. Comput. Informatics Syst.*, vol. 3, pp. 148–160, 2013.

[93] R. Householder, S. Arnold, and R. Green, “On Cloud-based Oversubscription,” *Int. J. Eng. Trends Technol.*, vol. 8, no. 8, pp. 425–431, 2014.

[94] Z. Xiao, W. Song, and Q. Chen, “Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1107–1117, Jun. 2013.

[95] T. Leesatapornwongsa and H. S. Gunawi, “The Case for Drill-Ready Cloud Computing,” in *Proceedings of the ACM Symposium on Cloud Computing - SOCC ’14*, 2014, pp. 1–8.

[96] S. P. Kavulya, K. Joshi, F. Di Giandomenico, and P. Narasimhan, “Failure Diagnosis of Complex Systems,” in *Resilience Assessment and Evaluation of Computing Systems*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 239–261.

[97] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King, “FTCloud: A Component Ranking Framework for Fault-Tolerant Cloud Applications,” in *2010 IEEE 21st International Symposium on Software Reliability Engineering*, 2010, pp. 398–407.

[98] P. Townend, P. Groth, and J. Xu, “A provenance-aware weighted fault tolerance scheme for service-based applications,” in *Proceedings - Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 2005, pp. 258–266.

[99] A. Bala and I. Chana, “Fault Tolerance- Challenges , Techniques and Implementation in Cloud Computing,” *Int. J. Comput. Sci.*, vol. 9, no. 1, pp. 288–293, 2012.

[100] Y. Zhang, A. Mandal, C. Koelbel, and K. Cooper, “Combined fault tolerance and scheduling techniques for workflow applications on computational grids,” in *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009.

[101] S. Jain and J. Chaudhary, “New fault tolerant scheduling algorithm implemented using check pointing in grid computing environment,” in *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*, 2014, pp. 393–396.

[102] D. Sun, G. Chang, C. Miao, and X. Wang, “Analyzing, modeling and evaluating dynamic adaptive fault tolerance strategies in cloud computing environments,” *J. Supercomput.*, vol. 66, no. 1, pp. 193–228, Oct. 2013.

[103] A. Avizienis, “The N-Version Approach to Fault-Tolerant Software,” *IEEE Trans. Softw. Eng.*, vol. SE-11, no. 12, pp. 1491–1501, Dec. 1985.

[104] M. N. Cheraghlou, A. Khadem-Zadeh, and M. Haghparast, “A survey of fault tolerance architecture in cloud computing,” 2016.

[105] R. Jhawar, V. Piuri, and M. Santambrogio, “Fault Tolerance Management in Cloud Computing: A System-Level Perspective,” *IEEE Syst. J.*, vol. 7, no. 2, pp. 288–297, Jun. 2013.

[106] R. Jhawar, V. Piuri, and M. Santambrogio, “A comprehensive conceptual system-level approach to fault tolerance in Cloud Computing,” in *2012 IEEE International Systems Conference SysCon 2012*, 2012, pp. 1–5.

[107] Y. Dai, Y. Xiang, and G. Zhang, “Self-healing and hybrid diagnosis in cloud computing,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5931 LNCS, Springer Berlin Heidelberg, 2009, pp. 45–56.

[108] P. Verissimo, A. Bessani, and M. Pasin, “The TClouds architecture: Open and resilient cloud-of-clouds computing,” in *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*, 2012, pp. 1–6.

[109] M. Harris, “Data Center Infrastructure Management,” in *Data Center Handbook*, 1st ed., USA: John Wiley & Sons, Inc, 2015, pp. 601–617.

[110] D. Cole, “Data Center Infrastructure Management (DCIM) Overview of DCIM,” *Data Cent. Knowl.*, 2012.

[111] X. Ouyang, P. Garraghan, C. Wang, P. Townend, and J. Xu, “An

Approach for Modeling and Ranking Node-Level Stragglers in Cloud Datacenters," in *2016 IEEE International Conference on Services Computing (SCC)*, 2016, pp. 673–680.

[112] S. Bhatia, A. Kumar, M. E. Fiuczynski, and L. Peterson, "Lightweight, High-Resolution Monitoring for Troubleshooting Production Systems," in *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*, 2008, pp. 103--116.

[113] U. Erlingsson, M. Peinado, S. Peter, and M. Budiu, "Fay : Extensible Distributed Tracing from Kernels to Clusters," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles - SOSP '11*, 2011, vol. 13, p. 311.

[114] A. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Commun. ACM*, vol. 55, no. 2, p. 55, Feb. 2012.

[115] X. Liu, Z. Guo, X. Wang, F. Chen, X. Lian, J. Tang, M. Wu, M. F. Kaashoek, and Z. Zhang, "D3S: Debugging Deployed Distributed Systems," in *Proc. of the NSDI - Conf. on Networked Systems Design and Implementation*, 2008, pp. 423–437.

[116] P. Reynolds, C. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat, "Pip: Detecting the Unexpected in Distributed Systems," in *Proc. of the NSDI - Conf. on Networked Systems Design and Implementation*, 2006, pp. 115–128.

[117] S. B. Davidson and J. Freire, "Provenance and Scientific Workflows: Challenges and Opportunities," in *Proceedings of the ACM SIGMOD international conference on Management of data*, 2008.

[118] D. Ghoshal and B. Plale, "Provenance from log files," in *Proceedings of the Joint EDBT/ICDT 2013 Workshops on - EDBT '13*, 2013, p. 290.

[119] A. Gehani and D. Tariq, "SPADE: Support for Provenance Auditing in Distributed Environments," in *Proceedings of the 13th International Middleware Conference*, 2012, pp. 101–120.

[120] S. Townend, P; Venters, CC; Lau, L; Djemame, K; Dimitrova, V; Marshall, A; Xu, J; Dibsdale, C; Taylor, N; Austin, J; McAvoy, J; Fletcher, M; Hobson, "Trusted Digital Spaces through Timely

Reliable and Personalised Provenance," in *15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, 2012, pp. 136–141.

[121] P. Townend, C. C. Venters, L. Lau, K. Djemame, V. Dimitrova, A. Marshall, J. Xu, C. Dibsdale, N. Taylor, J. Austin, J. McAvoy, M. Fletcher, and S. Hobson, "A Framework for Improving Trust in Dynamic Service-Oriented Systems," in *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, 2012, pp. 136–141.

[122] K.-K. Muniswamy-Reddy, M. Peter, and S. Margo, "Making a Cloud Provenance-Aware," in *1st Workshop on the Theory and Practice of Provenance (TaPP'09)*, 2009.

[123] S. M. S. Da Cruz, M. Manhaes, J. Zavaleta, and R. M. Costa, "Cirrus: Towards Business Provenance As-a-Service in the Cloud," *2012 IEEE 19th Int. Conf. Web Serv.*, no. i, pp. 668–669, Jun. 2012.

[124] P. Macko, M. Chiarini, and M. Seltzer, "Collecting provenance via the Xen hypervisor," in *Proceedings of 3rd USENIX Workshop on the Theory and Practice of Provenance (TaPP '11)*, 2011.

[125] S. Townend, P; Venters, CC; Lau, L; Djemame, K; Dimitrova, V; Marshall, A; Xu, J; Dibsdale, C; Taylor, N; Austin, J; McAvoy, J; Fletcher, M; Hobson, "Trusted Digital Spaces through Timely Reliable and Personalised Provenance," in *15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, 2012, pp. 136–141.

[126] P. L. Rupasinghe, H. H. Weerasena, and I. Murray, "Trustworthy provenance framework for document workflow provenance," in *International Conference on Computational Techniques in Information and Communication Technologies*, 2016, pp. 168–175.

[127] M. Ali and L. Moreau, "A provenance-aware policy language (cProvl) and a data traceability model (cProv) for the cloud," in *IEEE 3rd International Conference on Cloud and Green Computing*, 2013, pp. 479–486.

[128] Y. Li and O. Boucelma, "Provenance Monitoring in the Cloud," in *2013 IEEE Sixth International Conference on Cloud Computing*, 2013,

pp. 802–809.

[129] Y. Li and O. Boucelma, “A CPN Provenance Model of Workflow: Towards Diagnosis in the Cloud.,” *ADBIS (2)*, pp. 55–64, 2011.

[130] W. Zhou, S. Mapara, Y. Ren, Y. Li, A. Haeberlen, Z. Ives, B. T. Loo, and M. Sherr, “Distributed time-aware provenance,” *Proc. VLDB Endow.*, vol. 6, no. 2, pp. 49–60, Dec. 2012.

[131] J. Barillari, U. Braun, D. A. Holland, D. Maclean, M. Seltzer, and S. D. Holland, “Layering in Provenance-Aware Storage Systems,” *Aerosp. Eng.*, 2009.

[132] B. Lee, A. Awad, and M. Awad, “Towards Secure Provenance in the Cloud: A Survey,” *Proc. IEEE/ACM 8th Int. Conf. Util. Cloud Comput.*, pp. 577–582, 2015.

[133] C. H. Suen, R. K. L. Ko, Y. S. Tan, P. Jagadpramana, and B. S. Lee, “S2Logger: End-to-end data tracking mechanism for cloud data provenance,” *Proc. - 12th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. Trust. 2013*, pp. 594–602, 2013.

[134] G. Meera and G. Geethakumari, “A provenance auditing framework for cloud computing systems,” *2015 IEEE Int. Conf. Signal Process. Informatics, Commun. Energy Syst. SPICES 2015*, 2015.

[135] Y. Amanatullah, C. Lim, H. P. Ipung, and A. Juliandri, “Toward cloud computing reference architecture: Cloud service management perspective,” in *International Conference on ICT for Smart Society*, 2013, pp. 1–4.

[136] L. Moreau and P. Groth, “PROV-Overview: An Overview of the PROV Family of Documents,” *W3C Note*, no. April, pp. 1–9, 2013.

[137] P. Groth and L. Moreau, “Representing distributed systems using the Open Provenance Model,” *Futur. Gener. Comput. Syst.*, vol. 27, no. 6, pp. 757–765, Jun. 2011.

[138] A. Albatli, L. Lau, and J. Xu, “Application of PROV Model for Modeling a VM Overload Mitigating Strategy: Task Eviction,” in *Provenance Analytics Workshop*, 2014.

[139] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, “An Approach for Characterizing Workloads in Google Cloud to Derive Realistic Resource Utilization Models,” in *proceedings of the 7th IEEE*

International Symposium of Service-Oriented System Engineering, 2013.

[140] H. Aljahdali, A. Albatli, P. Garraghan, P. Townend, L. Lau, and J. Xu, “Multi-Tenancy in Cloud Computing,” in *Proceedings of the IEEE 8th International Symposium on Service Oriented System Engineering*, 2014, pp. 344–351.

[141] S. Braun, A. Schmidt, and A. Walter, “Ontology maturing: A collaborative web 2.0 approach to ontology engineering,” in *CEUR Workshop Proceedings*, 2007, vol. 273.

[142] B. Aksanli, J. Venkatesh, and Rosing, “Using Datacenter Simulation to Evaluate Green Energy Integration,” *Computer (Long. Beach. Calif).*, vol. 45, no. 9, pp. 56–64, Sep. 2012.

[143] C. Chang, L. He, N. Chaudhary, S. Fu, H. Chen, J. Sun, K. Li, Z. Fu, and M.-L. Xu, “Performance analysis and optimization for workflow authorization,” *Futur. Gener. Comput. Syst.*, vol. 67, pp. 194–205, 2017.

[144] S. Fu, L. He, X. Liao, and C. Huang, “Developing the Cloud-integrated data replication framework in decentralized online social networks,” *J. Comput. Syst. Sci.*, vol. 82, no. 1, pp. 113–129, 2016.

[145] Z. Du, L. He, Y. Chen, Y. Xiao, P. Gao, and T. Wang, “Robot Cloud: Bridging the power of robotics and cloud computing,” in *Future Generation Computer Systems*, 2016.

[146] U. Rahman, O. Hakeem, M. Raheem, K. Bilal, S. U. Khan, and L. T. Yang, “Nutshell: Cloud Simulation and Current Trends,” in *IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, 2015, pp. 77–86.

[147] S. Malekzai, D. Yildiz, and S. Karagol, “GreenCloud simulation QoSbox in cloud computing,” in *2016 24th Signal Processing and Communication Application Conference (SIU)*, 2016.

[148] M. Lacage and T. R. Henderson, “Yet another network simulator,” in *Proceeding from the 2006 workshop on ns-2: the IP network simulator - WNS2 ’06*, 2006, p. 12.

[149] B. Sotomayor, “The Haizea Manual,” *Science (80-.).*, 2009.

[150] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. R. Das, “MDCSim: A

multi-tier data center simulation, platform," in *2009 IEEE International Conference on Cluster Computing and Workshops*, 2009, pp. 1–9.

[151] G. G. Castañé, A. Núñez, and J. Carretero, "iCanCloud: A brief architecture overview," in *Proceedings of the 10th IEEE International Symposium on Parallel and Distributed Processing with Applications*, 2012, pp. 853–854.

[152] G. Keller, M. Tighe, H. Lutfiyya, and M. Bauer, "DCSim: A data centre simulation tool," *Integr. Netw. Manag. (IM 2013), 2013 IFIP/IEEE Int. Symp.*, pp. 1090–1091, 2013.

[153] Amazon Web Services Inc, "EC2 Instance Types – Amazon Web Services (AWS)," 2016. [Online]. Available: https://aws.amazon.com/ec2/instance-types/. [Accessed: 24-Oct-2016].

[154] P. Salot, "A Survey Of Various Scheduling Algorithm In Cloud Computing Environment," *Int. J. Res. Eng. Technol.*, vol. 2, no. 2, pp. 131–135, 2013.

[155] R. Birke and L. Y. Chen, "Managing Data Center Tickets : Prediction and Active Sizing," *2016 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Networks*, pp. 335–346, Jun. 2016.

[156] D. Thomas, A. Joiner, W. Lin, M. Lowry, and T. Pressburger, "The unique aspects of simulation verification and validation," in *2010 IEEE Aerospace Conference*, 2010, pp. 1–7.

[157] J. P. C. Kleijnen, "Verification and validation of simulation models," *Eur. J. Oper. Res.*, vol. 82, pp. 145–162, 1995.

[158] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, "A comparison of a graph database and a relational database," *Proc. 48th Annu. Southeast Reg. Conf. ACM SE 10*, p. 1, 2010.

[159] W.-T. Tsai, X. Sun, and J. Balasooriya, "Service-Oriented Cloud Computing Architecture," in *2010 Seventh International Conference on Information Technology: New Generations*, 2010, pp. 684–689.

[160] Z. Liu and S. Cho, "Characterizing Machines and Workloads on a Google Cluster," in *2012 41st International Conference on Parallel*

Processing Workshops, 2012, pp. 397–403.

[161] A. Albatli, D. McKee, P. Townend, L. Lau, and J. Xu, “PROV-TE: A Provenance-Driven Diagnostic Framework for Task Eviction in Data Centers,” in *Proceedings of the 3rd IEEE International Conference on Big Data Computing Service and Applications*, 2017.