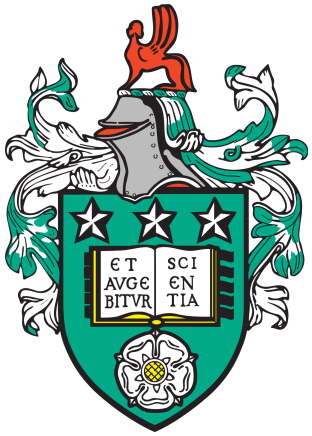
Enhancing Quality of Service in Cloud Computing Through Novel Resource Management

by

Django John Armstrong

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



The University of Leeds
School of Computing

December 2012

Abstract

Distributed Systems as an area of research has seen a gradual evolution over the last few decades fuelled by the application of new use cases to technological developments. Cloud Computing is one such paradigm that has evolved from the adoption of Utility Computing, Virtualization and Service Oriented Architectures. Cloud Computing can be distinguished from other distributed paradigms though the provisioning of resources, data and software to users on demand in a similar fashion to the services provided by the electric power industry. Commercial Cloud offerings are expected to meet the Quality of Service (QoS) requirements of a consumer via Service Level Agreements (SLA). In reality, Cloud providers rarely provide QoS beyond best effort as the intrinsic fault tolerant nature of currently deployed applications require little more. Nevertheless, with enhancements to QoS in Cloud Computing the range of deployable applications can be improved and thus advance the overall adoption of the paradigm.

This thesis tackles the shortcoming of QoS in Cloud Computing though novel enhancements to Cloud resource management. Since QoS is a broad subject area, the scope of research within has been narrowed down to two specific areas of interest: performance and scalability. In this thesis, the performance and scalability of Cloud technology are ascertained through performance evaluations on Hypervisor (such as XEN and KVM) and Cloud Infrastructure Managers (such as OpenNebula and Nimbus). Recommendations are made on how to resolve performance bottlenecks and on the suitability of certain technology for specific Cloud applications. Contextualisation and Re-contextualization mechanisms are introduced for self-configuring virtual Cloud resources at operation time while managing resources and software dependencies at the infrastructure and platform layer of the Cloud software stack. In addition, the thesis aims to improve the adoption of the Cloud by exploring novel techniques for composing, configuring and deploying Grid Middleware onto Cloud resources.

The core contributions of this thesis are as follows: i) A prototype software tool for the (re-)contextualization of Cloud applications, platforms, infrastructures and resource dependencies that enables improvements to performance, scalability and fault tolerance. ii) Performance results and recommendations on the topic of Virtual Machine (VM) image propagation delay in Cloud infrastructure technology, Paravirtualized block device drivers and VM image standards in Hypervisor technology, for the purpose of ascertaining current limitations in Cloud QoS. iii) A software prototype system of an interoperable self-configuring Virtual Grid infrastructure, deployable on to a range of Cloud providers, to enhance the QoS achievable by Grid applications.

Acknowledgements

The work I have performed in this thesis would have never have been possible without the guidance, dedication and support of my supervisor and mentor Dr. Karim Djemame, for which I am eternally grateful. His wealth of experience has provided me with encouragement and critical feedback throughout my PhD.

Additionally, I would like to thank the Collaborative Systems and Performance Research Group at the School of Computing and my OPTIMIS project colleagues for their friendship, support and enlightening discussions.

John Armstrong, (28/02/1929 - 30/12/2009)

My heart felt gratitude goes to my Mother and Brother for their continuing love and support, in addition to them enabling me to continue the family tradition of being the 3rd generation of Armstrong to graduate at the University Of Leeds. Finally and most importantly, without the commitment and motivation of my late father John Armstrong, I hesitate to think what state my life would be in now and dedicate this thesis to his memory.

Declaration

Some parts of the work presented in this thesis have been published in the following articles:

Book Chapters

A Performance Evaluation of Block I/O Paravirtualization & Virtual Machine Images. *D. Armstrong, K. Djemame*, Springer Book Chapter: CLOUD COMPUTING AND SERVICES SCIENCE, Ivan Ivanov, Marten Van Sinderen, Boris Shishkov (Eds.), May 2012

Journal Papers

Time/Cost Optimization in Grid Scheduling: Analytical Results and Practical Implications. *D. Armstrong, K. Djemame, N. V. Shakhlevich*, Submitted to EUROPEAN JOURNAL OF OPERATIONAL RESEARCH, July 2012

Legal Issues in Clouds: Towards a Risk Inventory. *K. Djemame, D. Armstrong, M. Kiran, M. Jiang. T, B. Barnitzke, M. Corrales, N. Forg*. To appear in the Philosophical Transactions of the Royal Society, 2012

Brokering of Risk-Aware Service Level Agreements in Grids. *K. Djemame, J. Padgett, I. Gourlay, and D. Armstrong*. CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE, Vol. 23, No. 7, May 2011

Performance Issues in Clouds: An Evaluation of Virtual Image Propagation and I/O Paravirtualization. *D. Armstrong and K. Djemame*. COMPUTER JOURNAL, 54 (6), February 2011, pp. 836-849

Refereed Conference Papers

Runtime Virtual Machine Recontextualization for Clouds. *D. Armstrong, D. Espling, J. Tordsson, K. Djemame, E. Elmroth*, Proceedings of the SEVENTH WORKSHOP ON VIRTUALIZATION IN HIGH-PERFORMANCE CLOUD COMPUTING VHPC'12, Euro-Par 2012, Rhodes Island, Greece, August 2012

Assuring Data Privacy in Cloud Transformations. *T. Kirkham, K. Djemame, M. Kiran, D. Armstrong and M. Jiang*, Proceedings of the ELEVENTH IEEE INTERNATIONAL CON-

FERENCE ON TRUST, SECURITY AND PRIVACY IN COMPUTING AND COMMUNICATIONS (IEEE TrustCom-12), Liverpool, UK, June 2012

Towards a Service Life Cycle-based Methodology for Risk Assessment in Cloud Computing. *K. Djemame, D. Armstrong, M. Kiran, M. Jiang*, Proceedings of the INTERNATIONAL CONFERENCE ON CLOUD AND GREEN COMPUTING (CGC'2011), Sydney, Australia, December 2011

Towards a Contextualization Solution for Cloud Platform Services. *D. Armstrong, K. Djemame, S. Nair, J. Tordsson, W. Ziegler*, Proceedings of the THIRD IEEE INTERNATIONAL CONFERENCE ON CLOUD COMPUTING TECHNOLOGY AND SCIENCE (CloudCom'2011), Athens, Greece, November 2011

A Risk Assessment Framework and Software Toolkit for Cloud Service Ecosystems. *K. Djemame, D. Armstrong, M. Kiran, and M. Jiang*. Proceedings of the SECOND INTERNATIONAL CONFERENCE ON CLOUD COMPUTING, GRIDs, and Virtualization, Rome, Italy, September 2011

Cultivating Cloud Computing: A Performance Evaluation of Virtual Image Propagation & I/O Paravirtualization. *D. Armstrong and K. Djemame*, Proceedings of the FIRST INTERNATIONAL CONFERENCE ON CLOUD COMPUTING AND SERVICES SCIENCE (CLOSER'2011), Noordwijkerhout, The Netherlands, May 2011

Towards Quality of Service in the Cloud. *D. Armstrong and K. Djemame*, Proceedings of the TWENTY-FIFTH UK PERFORMANCE ENGINEERING WORKSHOP, Leeds, UK, July 2009, pp. 226-240

Risk-Aware SLA Brokering using WS-Agreement. *J. Padgett, K. Djemame, I. Gourlay and D. Armstrong*. Proceedings of the TWENTY-THIRD IEEE INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION NETWORKING AND APPLICATIONS (AINA'2009), Bradford, UK, May 2009

Contents

List of Figures

List of Tables

List of Abbreviations

ABI Application Binary Interface

API Application Programming Interface

CCIF Cloud Computing Interoperability Forum

CFS Completely Fair Scheduler

CIFS Common Internet File System

CM Configuration Management

CO Cloud Optimizer

CRM Customer Relationship Management

DaaS Desktop As A Service

DFS Distributed File System

DHCP Dynamic Host Configuration Protocol

DMS Database Management System

DMTF Distributed Management Task Force

DNS Domain Name System

DRM Digital Rights Management

DRS Distributed Resource Scheduler

ERP Enterprise Resource Planning

Ext4 Fourth Extended File System

FTP File Transfer Protocol

FV Full Virtualization

GPL General Public License

GRAM Grid Resource Allocation and Management Protocol

GSI Grid Security Infrastructure

GSIFTP Grid Security Infrastructure File Transfer Protocol

GVC Globus Virtual Cluster

HaaS Hardware As A Service

HDD Hard Disk Drives

HDFS Hadoop Distributed File System

HPC High Performance Computing

HRM Human Resource Management

HTTP Hypertext Transfer Protocol

HVM Hardware Assisted Virtualization

IaaS Infrastructure as a Service

IdAM Identity and Access Management

IDS Intrusion Detections Systems

IP Infrastructure Provider

IP Internet Protocol

IPsec Internet Protocol Security

iSCSI Internet Small Computer System Interface

ISO International Organization for Standardization

KPI Key Performance Indicator

KVM Kernel Based Virtual Machine

LVM Logical Volume Management

MAC Media Access Control

MDS Monitoring and Discovery Service

MPI Message Passing Interface

NAS Network Attached Storage

NFS Network File System

NIC Network Interface Card

NIST National Institute of Standards and Technology

NNTP Network News Transfer Protocol

OCC Open Cloud Consortium

OGF Open Grid Forum

OGSA Open Grid Services Architecture

OGSA Open Grid Services Architecture

OMG The Object Management Group

OS Operating System

OVF Open Virtualization Format

P2P Peer to Peer

PaaS Platform as a Service

PBS Portable Batch System

PHP Hypertext Preprocessor

PKC Public Key Cryptography

PKI Public Key Infrastructure

POP3 Post Office Protocol

POSIX Portable Operating System Interface

PV Paravirtualization

QA Quality Assurance

QoS Quality of Service

RAID Redundant Array of Independent Disks

RDP Remote Desktop Protocol

REST Representational State Transfer

RISC Reduced instruction set computing

RPC Remote Procedure Call

RSL Resource Specification Language

SaaS Software as a Service

SAN Storage Area Network

SCP Secure Copy

SCSI Small Computer System Interface

SDO Service Deployment Optimizer

SECaaS Security As A Service

SLA Service Level Agreements

SM Service Manifest

SME Small and Medium Enterprise

SMTP Simple Mail Transfer Protocol

SNIA Storage Networking Industry Association

SOA Service Oriented Architecture

SOAP Simple Object Access Protocol

SP Service Provider

SSD Solid State Disk

SSH Secure Shell

STaaS Storage As a Service

TCP Transmission Control Protocol

UDDI Universal Description Discovery and Integration

UDP User Datagram Protocol

URI Uniform Resource Identifier

USB Universal Serial Bus

VIM Virtual Infrastructure Manager

VM Virtual Machine

VMI Virtual Machine Interface

VMM Virtual Machine Manager

VMX Virtual Machine Extension

VO Virtual Organisation

VPN Virtual Private Network

VPS Virtual Private Server

VW Virtual Workspace

WAN Wide Area Network

WS Web Service

WS-Agreement Web Services Agreement Specification

WSDL Web Service Description Language

WSRF Web Service Resource Framework

WWW World Wide Web

XML Extensible Markup Language

Chapter 1

Introduction

1.1 Research Motivation

Quality of Service (QoS) plays a critical role in the effective provisioning and reservation of resources within service oriented distributed systems and has been widely investigated in the now well established paradigm of Grid Computing [165]. The emergence of a new paradigm, Cloud Computing, continues the natural evolution of Distributed Systems (Figure 1.1) to cater for changes in application domains and system requirements.

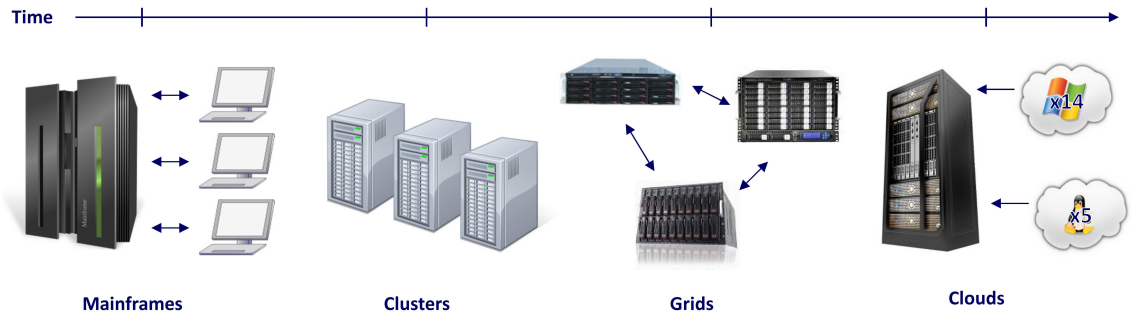


Figure 1.1. The Evolution of Distributed Systems

Virtualization of resources, a key technology underlying Cloud Computing, sets forth new challenges to be investigated within QoS and presents opportunities to apply the knowledge and lessons learnt from Grid Computing. QoS has been a topic of great interest in Distributed Computing paradigms, such as Grid Computing and High Performance

Computing [1, 45, 143]. The primary goal of this thesis is to address QoS specifically in the context of the nascent paradigm of Cloud Computing and its current best effort approaches to provisioning resources that are limiting its adoption. In reality, Cloud providers rarely provide QoS beyond best-effort, "*you get what you are given*", as the intrinsic fault tolerant nature of currently deployed Cloud applications require little more. Nevertheless, with enhancements to QoS in Cloud Computing the range of deployable application can be improved and thus advance the overall adoption of the paradigm.

QoS is a broad topic in Distributed Systems and is most often referred to as the resource reservation control mechanisms in place to guarantee a certain level of performance and availability of a service. The scope of the thesis is primarily concerned with the management and performance of resources such as processors, memory, storage and networks, in Cloud Computing. A defined QoS is not just limited to guarantees of performance and availability and can cover other aspects of service quality, which are outside the scope of the thesis, such as security and dependability. The problems surrounding resource provisioning and reservation are non-trivial for all but the most basic best effort guarantees and the problems behind resource capacity planning are often non-deterministic polynomial-time hard to solve [117, 234, 237].

The configuration of virtual Cloud resources and applications plays a role in the maximum obtainable QoS that can be achieved. Specifically, issues of performance, scalability and fault tolerance are effected if these elements of a Cloud are not configured correctly, this is discussed in detail as part of Chapter 6. An example of this could the misconfiguration of an application's software stack, preventing it from scaling across multiple virtual cloud resources.

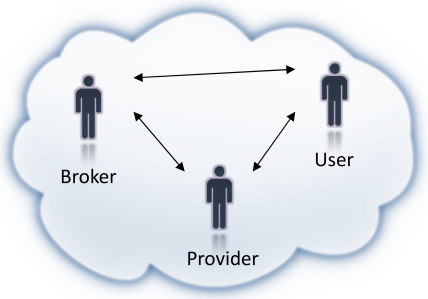


Figure 1.2. Quality of Service Actors

QoS provides a level of assurance that the resource requirements of an application are strictly supported. QoS models are associated with End-Users, Providers and often Brokers (Figure 1.2), involve resource capacity planning via the use of schedulers, load balancers and the utilisation of Service Level Agreements (SLA). SLAs provide a facility

to agree upon QoS between an End-User and Provider, defining the End-User's resource requirements and Provider's guarantees, thus assuring an End-User that they will receive the services they have paid for.

The work presented in this thesis is evaluated on a relatively small Cloud testbed. This places limitations on the usefulness of research results due to the inherent size of real Cloud infrastructures that can be in the order of hundreds of thousands of physical machines. No Cloud testbed will thus be a true representation of a real Cloud but the limitations can be overcome if the effects of small resource numbers are omitted from experimental variables within any evaluation. This issue is discussed in further detail within Section 5.4.1.

There are a number of tools available that facility the QoS of Cloud applications and more specifically scalability or *scale out* in Cloud specific terminology. An example of which is FlexiScale [76] that provide self-service provisioning of Cloud resources via a web-based control panel for end-users or a proprietary API for programmatic unattended scale out. Further details of the FlexiScale tool and others that enable scalability can be found in Chapter 2. Additionally, generic approaches to scalability in Clouds can be taken such as load balancing and redundancy but are not limited to Cloud based distributed systems.

1.2 Aim and Objectives

The aim of the thesis is to answer the following research question:

- How can QoS provisioning within Cloud Computing be improved through research into novel Resource Management?

This aim was achieved by investigating Resource Management and Quality of Service in Cloud Computing as part of four research activities that were identified as having the potential for creating novel solutions to problems directly related to Cloud Resource Management. The activities were as follows:

1. **Scalability in Clouds:** This research activity involved the investigation of scalability related issues in Cloud Computing and consisted of evaluating current technology and tools for performance bottlenecks, which could potentially limit the number of concurrent users able to access Cloud based services.

2. **Fault Tolerant in Clouds:** In this activity concepts such as availability and current architectures and tools were reviewed that hold relevance to the fault tolerant nature of Cloud resources.

3. **Interoperability in Clouds:** As part of this activity, the interoperability of different Cloud technologies designed to perform the same functionality in the software stack was analysed. Areas of improvement were distinguished that currently held back the adoption of Cloud Computing.

4. **Cloud Resources for the Grid:** This activity involved drawing parallels with existing Grid systems to find areas of common ground, where Clouds could leverage existing research results from the relatively mature paradigm of Grids Computing.

These activities proved to be highly relevant in investigating the research aim. Exploring these four activities in turn, provided the three research objectives of this thesis and were envisioned by identifying limitations in current research through the critical appraisal of the literature. Four activities were chosen to mitigate the possible risk of failure to deliver research results. Thus the Objectives of this research are as follows:

- **Objective 1 (O.1)** Ascertain and present recommendation on the performance, scalability and availability of Cloud technology, thus identifying areas where QoS can be improved. The recommendations hold pertinence to all actors of the Cloud ecosystem but are of most relevance to Cloud infrastructure providers and application developers wishing to minimise the economical impact that poor QoS provisioning can bring.

- **Objective 2 (O.2)** Enable enhancements to performance, scalability and fault tolerance in Cloud Computing improving QoS provisioning above current best-effort approaches.

- **Objective 3 (O.3)** Enable the use of interoperable Cloud Resources by the Grid community, improving the QoS obtainable by an application.

1.3 Methodology

In order to achieve the thesis objectives, a research methodology must be followed that exhibits scientific method and merit. There are two major research paradigms: quantitative and qualitative research [53]. The research performed in this thesis is based on the quantitative analysis of repeatable empirical experiments. From the literature there is evidence of the application of three research methodologies in Distributed Systems, they are:

- Prototyping [30]: Create a software prototype and validating its functionality against a set of requirements or existing solutions.
- Simulation [38]: Simulating a systems using a range of simulation methods and tools that can be validated against mathematical models or prototype implementations.
- Mathematical Modelling [22]: The formulation of mathematical models of systems, validated against prototypes or simulations.

In this thesis all three research methodologies are deployed but an emphasis is made on the use of prototyping and the evaluation of prototypes to fulfil and validate the objectives respectively. Mathematical modelling is employed only to facilitate the understanding of the reader. Prototyping is present in Chapter 6, simulation in Chapters 5, 6 & 7 and mathematical modelling in Section 7.3.3. Experiments on existing software and prototypes are developed to ascertaining the performance, scalability and availability of Cloud Computing environments and technology, using a wide variety of quantitative metrics and are gathered over multiple experimental iterations to reduce the effects of variance.

1.4 Main Contributions

The main contributions of this thesis were achieved by identifying and addressing the current limitations in Cloud Computing through cutting edge, contemporary research. These contributions that directly relate to the previously stated objectives are:

1. Experimental results on the performance and scalability of Cloud resource management software and Virtualization technology. This realises objective **O.1** and the results of this research can be found in Chapter 5.
2. A software prototype of a Contextualization tool and Recontextualization architecture for the purpose of configuring Cloud application's and their software stack, at deployment and operation time, for the purpose of enhancing performance, scalability and fault tolerance. For clarity, Contextualization at a high level is defined as the process of configuration for the purpose of providing unique identities to Cloud software components. Recontextualization is an extension of this concept where by Cloud software components are reconfigured and provided with a new identity. This realises objective **O.2**, the details of which can be found in Chapter 6.

3. A software prototype of a Virtual Grid architecture enabling the deployment of Grid applications onto public and private Cloud providers, while maintaining interoperability from the provider's resource management stack. This realises objective **O.3** and is discussed in Chapter 7.

A full list of all contributions of this thesis are presented in Chapter 8 in Section 8.2.

1.5 Thesis Overview

The remaining chapters of this thesis are as follows:

- **Chapter 2**: Introduces the concept of Cloud Computing, its classifications, deployment models and types. The landscape of Cloud architectures, both open source and commercial, are presented. Finally, the topic of Resource Management within Cloud Computing, including elasticity, scheduling and resource monitoring, are discussed. Additionally, the chapter presents the concept of Virtualization and virtual resource management in the context of Cloud Computing. The technology behind Virtualization, the Hypervisor, is discussed and classified with the techniques it uses to virtualise resources. Finally, a survey of contemporary Hypervisors, both open source and commercial, is presented including a discussion on their applicability to Cloud Computing.

- **Chapter 3**: Introduces Cloud application composition and the heritage it draws on from within the topic of Service Oriented Architectures. Cloud Engineering is discussed as a systematic approach to the creation and composition of Cloud applications and a number of issues to consider, when contemplating the use of Cloud Computing, are discussed.

- **Chapter 4**: Discusses the paradigm of Grid Computing, including architectural philosophies and applications that make use of it. In addition, Resource Management in Grids is discussed, including the relevance of Service Level Agreements and related to Cloud Computing. More so, a survey of Grid Middleware and the role that it plays within Grid Computing is presented. Finally, a comparison of Grids and Clouds is discussed.

- **Chapter 5**: Discusses the performance of Cloud infrastructure, identifying a number of issues. The concept of the Virtual Machine image is introduced, alongside the storage systems used to store them. The propagation of images within two

Cloud infrastructure management systems is discussed and compared. The topic of high performance Virtualization within Cloud infrastructure is discussed and two contemporary Hypervisors are compared. In addition the topic of Block I/O Paravirtualization is discussed, as a performance limiting factor of Cloud Computing. Finally, performance evaluations of Virtual Machine image formats, the resource provisioning & propagation delays in two Cloud management technologies and the overheads of virtual block I/O devices are presented, highlighting a number of areas for performance and scalability improvement in Cloud Computing.

- **Chapter 6**: Introduces the concept of contextualization as a mechanisms to enable the autonomous configuration of Cloud Computing software and discusses the landscape of tools available to help manage the complexity and scale of Cloud Computing systems. A generalised architecture for the contextualization of a range of Cloud software is presented. An implementation based on this architecture: the Contextualization Tools and the non-functional requirements it enables the fulfilment of are discussed. Additionally, the concept of Recontextualization, a mechanism and an architecture is presented as a way to enable the reconfiguration of a system after deployment. Finally, the performance of the Contextualization and Recontextualization implementations are evaluated and compared to related Cloud technologies.

- **Chapter 7**: Discuses the application of Cloud Computing resources in research through the execution of Grid Middleware on the Cloud. The concept of a Virtual Grid is discussed and an implementation, the Globus Virtual Cluster, presented. Additionally, the implications of running legacy applications on Clouds are discussed in light of a model for generating Cloud usage patterns. A legacy application used by Pathologist that can leverage the benefits of Cloud Computing is discussed, alongside a Cloud architecture for enhancing its scalability and performance. Finally, an evaluation of the Globus Virtual Cluster and the performance of the legacy Pathology application are presented.

- **Chapter 8**: Provides a summary of the work performed on a chapter by chapter basis, includes all the contributions of the thesis and presents an outline of some future work.

Chapter 2

Quality of Service in Cloud Computing

2.1 Introduction

In the most generalised context, Cloud Computing refers to the delivery of computing resources, such as compute and data resources, over a network to a remote user. As with any service, such as household utilities, QoS plays a critical role in ensuring that an end-user receives the service for which they have paid. QoS, for the purpose of this research is defined as a resource control mechanism that guarantees a certain level of performance and availability. There are a number of challenges facing QoS in Clouds. The two core challenges involve firstly, the guarantee of resource reservation by a binding agreement and secondly, the continued provisioning of a resource to specified requirements. In the context of Clouds, this translates to challenges in service provider interoperability where unification of resource control mechanisms and the resource types provisioned require standardisation and additionally in challenges a service provider must face with regards to managing their resources efficiently and in selecting an appropriate software stack to meet QoS requirements pertaining to the performance and availability of provided resources.

In this chapter the topic of Cloud Computing is introduced and a formal definition provided in light of the paradigm's heritage in Grid Computing. The relevance and scope of QoS applicable to the research performed in this thesis is discussed and three deployment models for the previsioning of Cloud resources are compared. A number of Cloud classifications are presented to clarify confusion around the types of Cloud available in the cur-

rent ecosystem. Taking these classifications into consideration, a number of open source Cloud architectures are presented and compared in a taxonomy in addition to several of the most prominent commercial Cloud offerings. Finally, the topic of resource management in Cloud Computing is introduced including scheduling algorithms and monitoring tools used to optimise the use of Cloud resources.

The remaining sections of this chapter are as follows: Section 2.2 discusses the paradigm shift to Cloud Computing and clarifies its ambiguity. Section 2.3 classifies Clouds into a number of genera given shared characteristics. Section 2.4 presents a number of open source Cloud architectures and Section 2.5 presents commercial Cloud offerings. Section 2.6 discusses the topic of resource management in Cloud Computing. Following this, Section 2.7 discusses the concept of Resource Virtualization its history and some areas of research. Section 2.8 presents the two classifications of Hypervisor technology. Furthermore, Section 2.9 discusses the five techniques that enable virtualization of computing resources, Section 2.10 introduces the concept of virtual machines images, the backend storage devices used to store them and the formats with which images can be stored in and finally, Section 2.11 presents a panorama of contemporary Hypervisors in active use within the IT industry.

2.2 Cloud Computing

Cloud Computing has been described as:

> "the next natural step in the evolution of on demand information technology services and products" [266]

within the field of Distributed Systems and draws heavily on the principles and paradigm of Grid Computing. As with any service, such as public utilities, guarantees need to be in place that pledge a certain level of performance and involves resources reservation control and monitoring mechanisms for service fulfilment. There has been much confusion over the term Cloud Computing due to its relative infancy within Computer Science, its extensive generalised use by industry and the lack of consensus on what a Cloud really is. Many definitions have been proposed and are often confused with the Grid paradigm.

Before the relevance of QoS within Cloud Computing can be considered, a concrete definition is essential in characterising current Cloud systems. This will facilitate in reducing the scope of research by excluding more generalised definitions of Clouds made by self professed computing experts such as:

> "using the Internet to allow people to access technology-enabled services." [86]

Being able to categorise a Cloud by its capabilities is key to formulating a concise definition and a simple taxonomy of Cloud Computing. A general consensus is held that Clouds fall into at least one of three fundamental models, dependent on the actors involved and the services provided [80, 104, 254]. High level definitions of these three types of model are:

- **Software as a Service (SaaS)**, defined as a provider that supplies remotely run software packages, via the Internet to consumers, on a utility based pricing model. A typical example application could be an on-line alternative to a word processor or spread sheet.
- **Platform as a Service (PaaS)**, defined as a provider that offers an additional layer of abstraction above a virtualised infrastructure. This provides a software platform that trades off restrictions in the type of software than can be deployed in exchange for built-in scalability.
- **Infrastructure as a Service (IaaS)**, defined as a provider that provisions compute and storage resource capacity via Virtualization allowing physical resources to be assigned and split dynamically.

These three models can be tiered on top of each other. For example a PaaS provider could make use of a third party IaaS provider or alternatively a PaaS provider could deploy and utilise their own IaaS or non-Cloud based infrastructure.

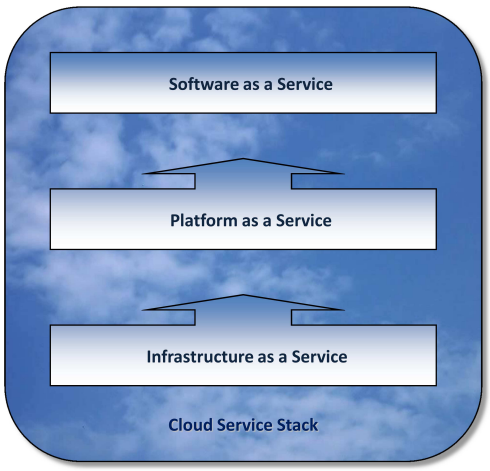


Figure 2.1. Cloud Service Stack

Previous tiers are obscured from the End-User and services are provided transparently, as illustrated in Figure 2.1. This allows for increased flexibility, the possibly of an open market and reductions in cost. Further detail on each model is provided in Section 2.3. Later in Section 2.4, example implementations of each type of system are discussed.

The evolution of Cloud Computing has its roots in multiple Internet related distributed system technologies and computing paradigms such as Cluster Computing, P2P, Service Computing, Utility Computing and most importantly Grid Computing.

QoS within Grids has been a major topic of interest and continues to be actively researched [45, 51, 143]. Research in Cloud Computing is reminiscent to the position that resource management and performance research was at in the early days of Grid Research [133, 172, 222], when issues were just becoming understood. Grid Computing has been defined as:

> "a system that: coordinates resources that are not subject to centralised control using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service" [81]

and was hailed as the next revolution in Computing Science after the creation of the Internet. Grids share many of the same goals as Cloud Computing. Thus the majority of the lessons already learnt within the research topic are highly relevant to Cloud Computing and can be utilised to increase the pace of development and the rate at which research into QoS in Cloud Computing can push the evolution of the paradigm.

The motivation behind research into Grid Computing materialised from the need to manage large scale resource intensive scientific applications across multiple administrative domains that required many more resources than which could be provided by a single institution. Cloud Computing shares this motivation but within a new context oriented towards business needs rather than those of academics, for the stipulation of reliable service chains rather than the previsioning of resources for batch oriented scientific applications. This difference in application domain pushed by industry, does not mean that the scientific community cannot leverage Cloud Computing, far from it, as illustrated by GridBatch [158] a programming model for using Clouds in large scale data intensive batch applications. There is much crossover between the two paradigms and many goals are shared.

Cloud Computing will be enabled even more so through the next generation of data centre technology. The current generation of data centres are already leaning heavily towards the Virtualization of compute and storage resources, the technological foundation of a Cloud, enabling the consolidation of proprietary servers running legacy software

and provides considerable reductions in the time and effort associated with administering hardware resources. This is achieved by the creation of Virtual Machines (VM) which run on large physical servers utilising the latest hardware technology, providing the benefits of both reduced hardware maintenance costs and the minimisation of lost revenue due to downtime. These benefits and the aforementioned differences in application domain requirements between Grids and Clouds have pushed Virtualization into the lime light as a new technological requirement of the Cloud paradigm.

This has only recently become feasible because of the performance enhancements that have been made to Virtualization hardware and software technology, which have improved the performance frontier of VMs to nearly that of the underlying hardware they run on. Near native performance of the virtualised resources exposed within a VM has been achieved through a reduction in the overheads associated with context switching physical resources between VMs and by taking advantage of the improvements in Virtualization enhancing hardware, touted by hardware vendors such as Intel and AMD.

Using the previously defined scope, the definition of a Cloud most relevant and appropriate to the research topic of QoS is:

> "Clouds are a large pool of easily usable and accessible virtualised resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to an optimum resource utilisation. This pool of resources is typically exploited by a pay-per-user model in which guarantees are offered by the Infrastructure Provider by means of customised SLA's." [254]

As it will become apparent later in this thesis, this definition of Cloud Computing is yet to be fully realised. The definition refers to a pay-per-user economic model that borrows heavily from the paradigm of Utility Computing. Utility computing is a:

> "service provisioning model, which provides adaptive, flexible and simple access to computing resources, enabling a pay-per-use model for computing similar to traditional utilities such as water or electricity." [160]

Research has already been carried out on the commercial benefits of Utility Computing within the Grid Economy [32, 65] and thus it is easy to envisage why such an economic model is important to Cloud providers and exploited within Cloud Computing heavily oriented towards business applications where maximising revenue is of primary concern.

2.2.1 Deployment Models

Cloud services can be used in a number of deployment configurations that have varying complexity, security implications and associated costs [104]:

- **Private**: Private Clouds are services or virtual infrastructure that is run and operated by a single institution. It can be managed by a third party and have associated physical infrastructure that is either on or off premise. It is comparable to building and managing traditional IT infrastructure and brings the benefits of enhanced security as no multitenancy is involved. A drawback of this approach is that the benefits of multitenancy, the economy of scale and the associated cost savings, are not realised. Many institutions choose to adopt Cloud Computing via this deployment model when data privacy and security are of the up most importance. Healthcare is a prime example where regulatory standards govern data privacy. Similarly, jurisdictional issues can provide motivation for this deployment model where different laws for managing data across countries can impeded the use of external Cloud providers.

- **Public**: In Public Clouds, services and infrastructure are provided to the general public and utilises a pay-per-use leasing policy. Many institutions choose this deployment model to accommodate spikes in service load where large numbers of users are concerned. Provisioning resources to service these spikes in load via traditional procurement methods would incur large investments and capital expenditure. In addition, these hardware resources would remain heavily underutilised at off peak times pertaining to inefficiencies. Thus institutions can use Public Clouds to reduce capital expenditure and operational IT costs.

- **Hybrid**: A Hybrid Cloud refers to the use of both Private and Public Clouds by an institution. This deployment model is opted for to provide fault tolerance and high availability. In addition, the model combines the benefits of enhanced scalability and reduced cost that public Clouds provide, with the security of private Clouds by enabling the deployment of certain sensitive applications internally.

- **Community**: In community Clouds several institutions share their infrastructure that implement the same terms of service and access policies. A good example is the sharing of IT resources by governments departments using Cloud technology. This deployment model can be likened to federated resources in Grids and with the formation of Virtual Organisation (VO). Federated Clouds are yet to be realised and

Community Clouds rely heavily on the use of homogeneous Cloud technologies due to a lack of standardisation.

In additional to federation, more advanced deployment models yet to be realised such as Brokering and Multi-Cloud scenarios are discussed later in Chapter 6.

2.3 Classifications

The following section provides a detailed landscape of all available Cloud service models at the time of writing. By classifying Cloud implementations and providing a taxonomy, distinguishing features sets can be compared and applicable uses cases identified, giving a baseline for further discussion.

2.3.1 Cloud Service Model

The term "*Cloud*" and its use as an umbrella term to market and explain the technical complexity of internet enabled services to technically adversed or illiterate end-users has complicated the standardisation of a definition. A consensus has slowly formed that the three layer model of SaaS, PaaS and IaaS can be used to classify most Cloud technology with the formalisation of a Cloud Computing definition by the National Institute of Standards and Technology (NIST) [104].

2.3.1.1 Infrastructure As A Service

IaaS provides the most basic services within the Cloud Service Model. Typically hardware resources such as computing power and data storage are compartmentalised and rented to end-users, in addition to other resources such as hardware firewalls and load balancers. In the context of commodity computer hardware, this is usually achieved via the use of Virtualization (explained in detail later in this chapter from Section 2.7) but is not a requirements (bare metal physical resources can be provisioned as a service). VMs, refereed to as Guests, run on top of Virtual Machine Managers (VMM) that are in-turn often refereed to as a Hypervisor. IaaS providers manage these pools of resources across entire datacenters, consolidating VM loads via resource schedulers that monitor resource usage patterns using a variety of monitoring tools. IaaS providers bill end-users on static resource allocations as well as on resources consumed on demand.

Typically, IaaS providers present the monitoring and management of there resources through a web based control panel for the manual provisioning of virtual resources by

end-users and via an API for automated provisioning. Automated provisioning of virtual resources using Key Performance Indicators (KPI) enable the unattended scaling of an application when KPIs thresholds are reached and acted upon to either provision more or less VMs.

In addition to providing virtual resources, many IaaS providers include VM image repositories containing ready made base images for the installation of software. An increasing number of IaaS providers are adding value to their existing services via the leasing of dedicated distributed storage systems, beyond basic block device storage, which is increasingly blurring the lines between infrastructure and platform middleware that one could argue are not traditional infrastructure services.

2.3.1.2 Platform As A Service

PaaS refers to the provisioning of software tools and APIs for consumption by end-users to create or run applications, in essence services that either integrate to make or host software applications. Services can be presented using SOAP and REST based interfaces, enabling service mashups, the composition of multiple Web Services to create a web based application. Potential downside to using PaaS offerings are vendor lock-in, if proprietary interfaces and development languages are used and a certain degree of infleaxibility, if a developer requires unavailable features or functionality. On the other hand, many PaaS offerings reduce the burden of implementing a number of non-functional requirements such as security, scalability and availability from the developer.

PaaS, when used in combination with IaaS and SaaS, provides middleware that acts as "glue" between the two layers of the software stack, similar to middleware in traditional Grid environments. This allows the application developed using PaaS to maintain a certain degree of separation from the underlying virtual infrastructure through a layer of abstraction. PaaS users normally pay on a per operation basis when integrating APIs with an application or on a per application basis when deploying an application into PaaS containers but billing methods can vary depending on the level of abstraction at which the PaaS is presented.

2.3.1.3 Software As A Service

SaaS is used to deliver a multitude of different services. In the context of enterprises, with which it is most often used, this can include but is not limited to the following business applications:

- Customer Relationship Management (CRM)

- Enterprise Resource Planning (ERP)
- Human Resource Management (HRM)

A potential critism of this term is that there is no concrete specification that a SaaS provider must utilise Cloud platform, virtualised infrastructure services or have multi-tenant users. Thus this classification of Cloud can be applied broadly to encompass any remotely hosted software package accessed by an end-user.

SaaS architectures often conform to using a single version and configuration of an application from a single point of presence. This can be contrasted with traditional software distribution, where multiple copies of potentially different software versions on dissimilar configurations of hardware and operating system can be found. The client side of a SaaS solution is often lightweight and presented using thin clients and web browsers. In addition, billing is usually based on the number of users accessing the service.

2.3.2 Other Cloud Types

Other terms using the "*As A Service*" moniker have since appeared to sub-categorise Cloud technology and define systems that blur the boundary between the three layer model.

2.3.2.1 Storage As A Service

Storage As a Service (STaaS) is a model for storing data remotely, off-site and has gained in popularity as data storage requirements in computing have exploded. Recent data storage growth rate studies have shown that an exponential rate of increase is expected [72,85] with the world crossing the zettabyte (10^{21}) storage barrier in 2010.

StaaS providers provision Hard Disk Drive (HDD) resources to end-uses using PaaS or SaaS solutions. The individual data resources are amalgamated and presented as an infinite resource using distributed file systems. Users of such services are billed on the amount of data stored. The economy of scale that a provider enables, makes off-site data storage for backup purposes an attractive alternative for individuals and enterprises a like. A potential downside to using StaaS is that there is often a cost associated with uploading the data to the provider. In addition, the asymmetry of many internet connections, where upload bandwidth is often an order of magnitude smaller than download, reduces the feasibility of the services for some users. QoS and privacy has also limited adoption, with the majority of providers making no promise that the data will be stored reliably or securely.

2.3.2.2 Hardware As A Service

Hardware As A Service (HaaS) is the resale of dedicated physical computing resources and functions similarly to IaaS. This can range from individual servers to colocation services that provide server rack space, network connectivity and power. The HaaS model is a new umbrella term for publicly available IT services that were available before the concept of Cloud Computing was popularised. HaaS can provide a cost saving alternative to traditional visualised Cloud services if rapid scaling is not required or if the performance overheads of Virtualization technology and the contention caused by multitenacy are likely to have an adverse effect on QoS.

2.3.2.3 Desktop As A Service

Desktop As A Service (DaaS) is a model for providing desktop environments to a large number of users from a remote location. It uses virtualization to provision many encapsulated desktops on a single physical machine to many thin clients. Typically, DaaS is sold on a per user license that includes the licensing costs of the operating system and installed software applications.

2.3.2.4 Security As A Service

Security As A Service (SECaaS) is a model for previsioning large scale security services such as Virtual Private Networks (VPN), anti-virus and intrusion detection systems on a subscription basis. Privacy concerns have seen a recent rise in the popularity of secure anonymity based services with Internet Service Providers (ISP) forced to block illegal file sharing websites, seen as the first steps to wider monitoring, tracking and censorship of the internet.[1].

2.4 Open Source Architectures

Advanced knowledge of Cloud Computing technology provides insight into QoS limitations. This subsection discusses the architectures of popular open source Cloud software and the use-cases they are best suited towards. These technologies fall into the previously discussed IaaS and PaaS models.

[1]BBC: The Pirate Bay must be blocked by UK ISPs, court rules `http://www.bbc.co.uk/news/technology-17894176`

2.4.1 Infrastructure as a Service

In this section four software projects that were originally design to meet the needs of an IaaS providers are discussed.

2.4.1.1 OpenNebula

The first Cloud architecture evaluated, OpenNebula [188], is based on the research efforts of the Reservoir Project [207]. Reservoir was a European lead research initiative into virtualised infrastructure and Cloud Computing.

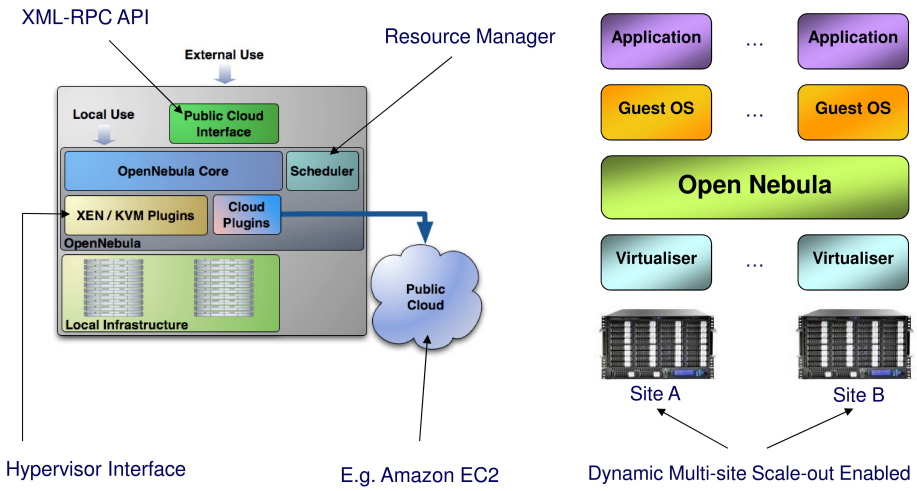


Figure 2.2. OpenNebula a) IaaS Architecture [188]. b) Software Stack.

The Reservoir Project primary deliverable was a complete definition of a reference IaaS architecture, depicted in Figure 2.2(a), built on open standards to provide a framework for the delivery of scalable, flexible and dependable services. The project aimed to develop key technologies enabling the migration of VMs across network and storage boundaries (illustrated by Figure 2.2(b)), algorithms for the effective allocation of resources in conformance to SLA requirements and a testbed to benchmark the performance of the architecture in industrial and commercial uses cases.

OpenNebula has become a prominent implementation of an open sources IaaS, due in part to its support from academic research groups actively publishing research on its applications. It is a feature rich system with a concrete direction for future development. The OpenNebula architecture has been designed with modularity in mind, making extensions easier to create by external developers and has been tested on large scales with many thousands of nodes.

The core of the architecture is comprised of a number of components that control and monitor resources the like of VMs, virtual networks, storage and physical host machines. These components are:

- **Request Manager**: This component is responsible for handling client requests. The component exposes an XML-RPC interfaces with a number of methods for managing resources.
- **Virtual Machine Manager**: This component manages and monitors VM resources and the operations it provides are abstracted from the underlying Hypervisor technology used.
- **Transfer Manager**: This component hands the transfer of images between host machines and includes the transfer of images to and from an image repository.
- **Virtual Host Manager**: This component handles assignment of Internet Protocol (IP) and Media Access Control (MAC) addresses, enabling the creation of virtual networks by tracking IP leases.
- **Host Manager**: The monitoring of physical machines is managed by this component via suitable drivers that can be extended to monitor any host attribute.
- **Database**: Persistence if enabled via an SQLite database for storing OpenNebula internal data structures that represent the state of a resource pool. This component enables reliability in the case of a failure or system restart.

Resources scheduling is performed by a scheduler component that is independent of the rest of the architecture and communicates with the core components using XML based remote procedure calls. This enables decoupling so that the scheduling algorithms and mechanisms used can be tailored or changed to a specific provider's needs. An example of a scheduler that can be plugged in is discussed later in Section 2.6.2.2. The default scheduler distrusted with OpenNebula enables the definition of resources and load aware policies.

2.4.1.2 Eucalyptus

The next IaaS Cloud architecture to be critically appraised is Eucalyptus [71], an IaaS system with the aim of creating:

> "an open-source infrastructure architected specifically to support Cloud Computing research and infrastructure development." [181]

The system architecture comprises of a Cloud Controller component responsible for processing incoming user requests, manipulating the Cloud fabric and processing SLAs in company with a Client Interface that utilises Internet standard protocols such as HTTP, XML and SOAP. In addition a Storage Controller component enables the management of block storage devices that can be dynamically attached to a VM. One of the primary benefits of using Eucalyptus over other IaaS implementations is that it maintains user facing compatibility with Amazon EC2 (compute) and S3 (storage) [5] service APIs but has also limited novelty within the project.

2.4.1.3 CloudStack

CloudStack [49] is a mature (available since 2009) open source IaaS solution that is governed by the Apache Software Foundation [10] and currently supported by Citrix. It has been designed with scalability and centralised management in mind. All major Hypervisors are supported including support for non-visualised bare-metal servers, differentiating this solution from others.

The architecture of a CloudStack Cloud is strictly hierarchical in structure enabling the solution to scale to many thousands of physical servers from a single management interface. The architecture of CloudStack is based on three tiers:

- **Zone**: The largest organisational unit within CloudStack, typically a datacenter will contain a single Zone. This enables geographical zoning of resources and data to be placed at a specific location to for compliance with an institution's policies. A Zone consist of at least one "Pod" and Secondary Storage component.
  - **Secondary Storage**: Is used to store VM templates and enable data replication between Zones, providing a common storage platform throughout a Cloud. The components makes use of the Network File System (NFS) protocol to ensure network access by any host within a Zone.
- **Pod**: Is hardware configured to form a "Cluster" of resources. A Pod is typically comprised of a rack of servers and shared networking hardware. Pods are not visible to end users.
- **Cluster**: Is a group of host machines running identical Hypervisors. Each Cluster has a dedicated Primary Storage device in which VM instances are hosted. A Cluster provides high availability and load balancing features.

– **Primary Storage**: Is built using high performance dedicated hardware to accommodate concurrent access by multiple VM instances. The component is designed to provision an instance with storage using standard compliant iSCSI and NFS protocols.

CloudStack is written in Java and provides a Management Server component backed with a database to store Cloud state. This enables the management of resources via a web interface, command line interface or REST based API.

2.4.1.4 Nimbus

The final IaaS open source architecture discussed is Nimbus [177]. This project provides a toolkit for the creation of IaaS services that are tailored to the needs of science but with many non-science use cases still supported.

The architecture of Nimbus is comprised of a Workspace Service component that provides standalone VM management with support for two remote protocol frontends. The first is the Web Service Resource Framework (WSRF) [276] frontend and an incomplete implementation of the Amazon EC2 interface that supports a number of EC2 management operations. A Workspace Control component acts as an agent for the management of individual host nodes and implements support for a number of Hypervisors and network configurations. In addition, a Metadata Server component enables the querying of VM state information.

Architectural components are loosely coupled together, self-contained and can be composed in a number of ways to enable different resources clustering strategies. In addition to providing IaaS functionality, Nimbus is tailored to support the academic community with support for many Grid standards. This is discussed in further detail as part of Chapter 7.

2.4.2 Platform as a Service

In the following section three open source PaaS toolkits and their associated architectures are discussed.

2.4.2.1 OPTIMIS Toolkit

The OPTIMIS Toolkit [74, 191] is a deliverable of the Optimized Infrastructure Services project, aka 'OPTIMIS', a three-year, 10.5m Euro research and development project, se-

lected under the 'Software and Service Architectures & Infrastructures' track of the EU's FP7 framework program.

Management actions in the toolkit are harmonized by ubiquitous policies that take into account *TREC*: *'T'*rust and *'R'*isk assessment to comply with *'E'*cological and *'C'*ost objectives without compromising on operational efficiency. The tools enable developers to enhance services with non-functional requirements regarding the allocation of data and virtual resources, as well as aspects related to performance such as elasticity, energy consumption, risk, cost, and trust. The toolkit incorporates risk assessment in all phases of the service life cycle and uses trust assessment tools to improve decision making.

The architecture of the OPTIMIS toolkit is comprised of a number of components that can be combined together to form a PaaS provider. In addition, a subset of the tools can be used to enable IaaS functionality. The OPTIMIS Toolkit is discussed further in Chapter 6. The high level core components that form the architecture of the toolkit and accommodate non-functional requirements are:

- **Monitoring Infrastructure**: This component is comprised of a number of tools that collect and monitor service performance indicators including physical resource energy efficiency. This monitored data is used as part of the TREC assessments.
- **Security Framework**: This component secures aspects of the toolkit including communication channels and data.
- **TREC Components**:
  - **Trust**: This component is responsible for accessing the reputation of a provider given its past history.
  - **Risk**: This component provides a number of risk assessments related to the failure of a service given a SLA and agreed level of QoS.
  - **Eco-efficiency**: This component assess energy consumption and carbon emission of a given Cloud infrastructure.
  - **Cost**: This component is responsible for assessing and predicting the costs associated with deploying and operating a service given a defined QoS

In addition, core functional requirements are provided for by the following components:

- **Fault Tolerance Engine**: The fault tolerance component enables self-healing infrastructure for the purpose of maintaining QoS. The component make use of monitoring data to detect failed virtual resources and reactively restarts them elsewhere.

In addition, proactive fault tolerance can be used to migrate virtual machines if the risk of hardware failure is high on a given physical machine.

- **Data Manager**: This components provides a distributed data repository for the storage of VM images and application data via a Java API.
- **VM Manager**: This component provides basic VM management operations such as stop, start, restart and provides a layer of abstraction above the underlying IaaS and Virtualization layer enabling a certain degree of interoperability.
- **Integrated Development Environment**: This component provides a programming model and graphical interface for the development and composition of a Cloud application and makes use of a number of PaaS services provide by the toolkit.
- **Service Manager**: The Service Manager component aims to optimise the management of an application during operation given a number of business level objectives.
- **Virtual Machine Contextualizer**: Is used to configure OPTIMIS platform level tools and application dependencies. Its architecture is described in detail as part of Section 6.3.

2.4.2.2 OpenStack

OpenStack [189] is a collaboration effort between a number of global developers and Cloud Computing technologists that aims to produce an open source Cloud platform for public and private deployment models. The project was founded by Rackspace (See Section 2.5.1.5 for more details) in collaboration with NASA and has grown to encompass a global software community working together on a standard and highly scalable open source Cloud software stack.

The OpenStack architecture is comprised of three core projects that provide functionality for the creation of private Clouds. These projects are Compute, Object Storage and Image Service that manage VMs, data storage and VM images respectively.

The Compute project provides access to virtual resources via an API that provides:

- Basic VM management (Start, Stop, Reboot, Resize).
- Resource quota management.
- VM image caching on compute nodes for faster provisioning times.
- Role based access control and identity management.

- Distributed and asynchronous access to virtual resources.

The Storage project provides reliable and scalable object (key:value) based storage and is not a traditional file system but rather presented as a distributed storage system through an API similar to Amazon's S3 PaaS. The API is REST based or consumed by a binding language that consumes the REST API and is thus not a POSIX (a set of standards for defining compatible interfaces to storage across operating systems) compliant file system and cannot be mounted at the operating system level. This limits the software's applicable uses cases to storing and retrieving files such as the archiving of media files and loading of VM images.

The Image Service provides the discovery, registration and delivery of VM images with the ability to create snapshots. The backend of the Image Service can be used with a variety of storage APIs including the object storage provided by the OpenStack Object Storage project. Image Service enables the use of template images to spawn multiple VM instances from a base image. Most VM image formats are supported.

2.4.2.3 Hadoop

Hadoop [113, 272] is a PaaS framework written in Java that supports data intensive applications. It enables the use of thousands of machines to operate on petabyte datasets. Hadoop makes use of the MapReduce [56] paradigm popularised by Google. MapReduce is the application of an old idea in a new context, the use of algorithmic skeletons [50] in Cloud Computing. Algorithmic skeletons are high level parallelism patterns that hide the complexity of developing parallel and distributed applications from the developer. MapReduce is very similar in concept to the algorithmic skeleton of `fork` and `join` where jobs are forked from a parent process and join back after completion, after which the main body of execution continues.

The core architectural components of Hadoop are the MapReduce Engine and Hadoop Distributed File System (HDFS) [226]. MapReduce is a programming model for the processing of data that was inspired by functional programming and the commonly used `map` and `reduce` functions. The `map` step takes data as input, divides the data into smaller subsets and distributes the subsets amongst a cluster of worker nodes which return the processed subset of data to the master node on completion. The `reduce` step involves combining the processed subsets of data returned from the slave nodes into a usable format or solution to a specific problem. MapReduce can make use of the locality of data where by the processing of data is performed near storage resources to decrease the overheads associated with transmitting data over a network. The Hadoop Distributed File System

(HDFS) is a locality aware file system written in Java that handle access to large files for worker nodes in the Hadoop platform. The file system makes use of TCP/IP and Remote Procedure Calls (RPC) for communication, replicating data across multiple hosts for redundancy. HDFS is not fully POSIX compliant and thus cannot be mounted at the operating system level but does provide increased throughput and performance for its target goal of supporting Hadoop applications.

The MapReduce Engine is comprised of two sub-components the JobTracker and TaskNode. The Jobtracker schedules and stores the location of running MapReduce jobs. The TaskNode runs jobs allocated to it from the JobTracker. HDFS is composed of another two sub-components, the NameNode and DataNode. The NameNode is responsible for storing file metadata including locality information and file attributes. The DataNode stores file data and can communicate with other DataNodes to rebalance the distribution of data within a cluster to improve performance so that the data is as close as possible to the computational resources with which it is to be processed. A basic Hadoop cluster consists of a master node and multiple worker nodes with the following structure:

- Master Node
  - JobTracker
  - TaskTracker
  - NameNode
  - DataNode
- Slave Node
  - TaskTracker
  - DataNode

The role of the Master node is to schedule tasks on to slave resources using the map reduce paradigm. In addition, the Master tracks the progress of Slave nodes.

2.4.3 Other Architectures

The previously discussed open source Cloud architectures are by no means an exhaustive list and there are other less infamous IaaS and PaaS solutions available. These are covered in numerous survey papers [170, 173, 194, 208].

2.5 Commercial Clouds

Understanding the specific problems surrounding QoS in commercial Clouds is a difficult task as the services provided are not transparent, the End-User has no idea of the underlying implementation. This is similar to the approach taken in Service Oriented Architectures (SOA), such as with Web Service Mash-Ups, where only the functionality of service is exposed by a defined interface.

2.5.1 Commercial Providers

An overview of commercial Cloud vendors, the technology they have in place and the state of their QoS provisioning is essential for the priorities of academic research to be in sink with the needs of businesses and for research in to QoS to be of real world intrinsic value.

The majority of major commercial Cloud vendors provide best effort QoS provisioning only and the most basic of guarantees on the availability and performance of resources. This provides primary motivation to research QoS in the context of Cloud Computing, furthering the adoption of the paradigm. Research has already made some strides into evaluating the variance in allocated resources provided by the Amazon Cloud in [132] and continues to be a research topic of interest.

In addition, due to the closed source proprietary nature of these commercial Clouds, limitations are present concerning interoperability. Unlike Grids the nature of Clouds are very much orientated towards providing services "behind closed doors". This is resulting in an emergent area of research investigating the development of Cloud standards to enable the sharing of Cloud resources outside administrative and organisational boundaries, standards that could also encompass QoS and provide the basis for Cloud brokering systems which are not possible without standard interfaces to communicate with and descriptive languages to define Cloud services.

In this subsection several commercial adopters of Cloud technology are discussed including the services and software products they provide that are guiding the direction research is taking within the paradigm of Cloud Computing.

2.5.1.1 Amazon

Amazon the first company to supply Cloud infrastructure services via its Amazon Web Service [5] products in early 2006, provides a PaaS architecture on a pay per use financial model. The architecture is marketed as two individual products: the Amazon Elastic

Compute Cloud (Amazon EC2); and the Amazon Simple Storage Service (Amazon S3). These products provide a set of well defined APIs, that have become widely adopted as standards in many open source Cloud architectures, such as Enomalism [69], Eucalyptus [71] and OpenNebula [188]. These projects provide interfaces compatible with Amazon's services to enable on demand scale out of service workloads to supplement local resources and satisfy peak or fluctuating demands.

2.5.1.2 Google

Another contender positioning themselves as a provider of Cloud services is Google. Google provides SaaS via its Google Apps [95] software and a PaaS via its Google App Engine [96]. The Google App Engine provides the architecture that Google Apps run on and promises transparent scalability on a pay-per-use financial model. The Google App Engine is limited to a subset of the Python API and provides a proprietary data storage query language limiting its applications.

2.5.1.3 IBM

IBM has released literature on its vision of Cloud Computing [29] and provides a PaaS based around the API's created by Amazon, known as IBM's Research Compute Cloud [124]. IBM also supplies enterprise Cloud Computing solutions in the form of a Cloud Service know as IBM Computing on Demand [125].

2.5.1.4 Microsoft

Microsoft provides Cloud services using the Azure Services Platform [20], a PaaS operating system, which integrates many of Microsoft's current proprietary software packages into one, via a layer of middleware, which can be utilised by licensed Cloud vendors and is marketed as an all-in-one Cloud software solution.

2.5.1.5 Rackspace

Rackspace [201] is an IT hosting company based in the USA that provides managed Cloud IaaS around the world. Its client base covers 40% of the Fortune 100 publicly trading corporations in America. Recently, Rackspace have announced its plans to provision its Cloud resources via the use of OpenStack and is in the process of integrating OpenStack Compute into its existing underlying technology [202].

2.5.1.6 SalesForce

SalesForce [220] provides CRM software to enterprise businesses. With the advent of Cloud Computing, SalesForce is provisioning its software via pay-per-use SaaS model and reselling its spare resource capacity via a PaaS solution that enables external developers to create add-on applications that integrate with its existing SaaS products. In addition, some of SalesForce's CRM products can be run externally on Amazon Web Services.

2.5.1.7 Flexiscale

FlexiScale [76] is a European based IaaS solution that makes use of the Flexiant Cloud Orchestrator software (see Section 2.5.2.4 for details). It provides self-service provisioning of resources using a web-based control panel or proprietary API and is backed by a virtualised storage backend. QoS is maintained via fully automated hardware failure recovery and instantaneous backup and restore capabilities via disk snapshots.

2.5.2 Commercial Software Stacks

In addition to the open source software available for use in creating a Cloud, there are a number of commercially available alternatives. This section of the thesis provides an overview of these software solutions.

2.5.2.1 VMware vCloud

The VMware vCloud Suit [264] is a IaaS solution that combines many of the existing software offerings of VMware including its commercial Hypervisor technology. vCloud has a number of components that provide features and functionality a potential provider can use. The core components are:

- **vSphere**: Enables the management of virtualised infrastructure using policies for the automated provisioning of compute resources.
- **vCloud Director**: Provides software to create a complete virtual datacenter including virtualised compute, networking, storage and security resources.
- **vCloud Connector**: Enables hybrid Clouds via the dynamic transfer of workloads between private and public Cloud providers.
- **vCloud Networking and Security**: A software defined networking and security solution for providing virtual firewalls, VPN, load balances and virtual LANs.

- **vCenter Site Revovery Manager**: A software solution for protecting Cloud applications from resources failures, providing VM migration and replication as fault tolerance mechanisms in addition to non-disruptive testing and centralised recover plans for defining virtual machine dependencies.

2.5.2.2 Enomaly

The second commercial Cloud software stack discussed is Enomaly [69], another IaaS system that presents an organisation with the capability to manage virtual infrastructure, VM images and provides fine grained security and user management. In addition, the solution enables the creation of VM images and supports multiple Hypervisors. Automation is enabled via a REST based API and manual control over resources is provided by a customer and administrative web based control panel.

2.5.2.3 CA AppLogic

AppLogic [13], a recent acquisition by CA Technologies, is a "turnkey" style software solution for the provisioning of IaaS. Compared to other IaaS solutions, AppLogic supports complex application deployments including software firewalls Virtual Private Networks (VPN) and load balancers that are configured via a drag and drop web based user interface.

2.5.2.4 Flexiant Cloud Orchestration

Flexiant [75] is a UK based Small and Medium Enterprise (SME) that provides IaaS Cloud software to a European consumer base and is touted as providing the worlds second Cloud Computing platform after Amazon and Europe's first. Its software solution, named Flexiant Cloud Orchestration, provides role based access control and metering mechanisms in addition to an intelligent workload placement and virtual resource scheduler that can be tuned to ensure reliability, resilience and performance of an application in a dynamic environment.

2.5.3 End-User Applications

A number of commercial entities, in addition to those provisioning Cloud services to other businesses, provide Cloud resources to End-Users in the form of SaaS. The following presents a non-exhaustive list of examples: Cloud based storage, office suites and Cloud gaming services.

2.5.3.1 Cloud Storage

A number of providers have evolved to make use of the Cloud Computing paradigm to service users in the home and there ever expanding storage needs. This has been enabled by the economy of scale that Cloud Computing provides, simplified scalability and by the expanding availability of high bandwidth internet connection in the home.

Dropbox [64] was one of the first Cloud storage provider to provision storage resources for remote file backup and enable the synchronization of files between computers. It follows a business model where users can create a free account with limited storage and pay on a per use basis for addition capacity. Dropbox make extensive use of Amazon's S3 storage system to store user's files. There have been recent concerns over the privacy and security of Cloud Storage services with breaches in security receiving wide spread media coverage[2].

Recently other more established organisations in the IT industry have entered this market as estimated revenue has increased. Direct competitors to the Dropbox service include the likes of Google, Apple and Microsoft with the products: Google Drive [98], Apple iCloud [12] and Microsoft SkyDrive [168] respectively.

2.5.3.2 Office Suites

With the rise of Web 2.0 and websites that enable interactive collaborative environments in addition to user generated content, the feasibility of online office suites provided by the SaaS model has been realised. Online office suites provide web based document creation and editing applications with document storage and sharing facilities. Typical applications include email, spreadsheet, shared calendars and a word processor. Google Docs [97] is one such example where users are able to edit documents online free of charge using an ad supported revenue model. Office 365 [167] is Microsoft's answer to Google Docs that maintains full compatibility with there existing desktop Office solution and uses a monthly subscription fee on a per user basis.

2.5.3.3 Cloud Gaming

Cloud Gaming is the on-demand streaming of computer games over the internet to client machine, where by the actual game is stored and executed on a remote server. There are several benefits to this type of online gaming, the hardware on which the client runs needs only minimal resources to display the streamed frame buffer and send input to a remote

[2]BBC News: Dropbox details security breach that caused spam attack. `http://www.bbc.co.uk/news/technology-19079353`

location. Moreover, the use of invasive Digital Rights Management (DRM) software to protect a title from piracy is no longer required as game code is executed in a remote protected environment. There are however technical limitations to Cloud Gaming due to bandwidth and latency constraints that limit its use in rural areas. Examples of platforms that support Cloud gaming are OnLive [185] and StreamMyGame [240].

2.6 Resource Management

In this section of the thesis the topic of resource management in Cloud Computing is discussed, including a clarification of what elasticity means in the context of a Cloud. Scheduling of virtual resource and the algorithms used in IaaS implementations, one open source, another that accommodates advanced reservation and a commercial offering are discussed, in addition to monitoring tools used to detect environmental changes that provides input to Cloud resource schedulers.

Resources management in distributed systems, in the most generalised of contexts, refers to the efficient allocation of workload to a shared computing resource. This is achieved by setting a goal, such as maximising resource utility or workload throughput, given a set of constraints, often technological and economical. Resource management involves the following:

- **Characterisation**: Knowledge acquisition and understanding of the system workload and its resource requirements.
- **Allocation**: The distribution of workload to resources across competing tasks or services.
- **Adaptation**: The accommodation of system and environmental changes such as failures and changes in workload.

These three aspects of resource management are only possible through the monitoring of system resources. In Cloud Computing resource management differs to traditional distributed systems such as Grids and clusters due to Virtualization. VMs are multiplexed between the resources of a physical machine enabling multitenancy and resource sharing. In addition, Virtualization provides a layer of abstraction above the physical resource, enabling a two tier approach to resource management in a IaaS provider. An application running on Cloud infrastructure is thus unaware of the underlying physical environment with which it is executing. This make the characterisation of workload difficult as the infrastructure layer in a Cloud is unaware of the attributes associated with an application

running inside a VM and is an active area of research [19]. The goal of an IaaS provider is to maximise revenue which involves the minimising of power consumption and maximization of resource utility via the consolidation of VMs onto physical machines.

In Cloud Computing monitoring tools are used to characterise the current workload that a VM is placing on a physical machine. When additional VMs are brought online the current state of the Cloud is evaluated and the VMs are allocated to a suitably underutilised resource that meets the QoS requirements. In the event of hardware failure, fault tolerance is achieved via the migration of a VM to a functional resources. The live migration of running VMs also play a role in adapting to changes in demand. Workloads in Clouds do not remain static and change during periods of peak user activity. Thus a VM that is using many resources could have a potential QoS impact on other VMs running in the same physical machine and should be migrated to a resource with appropriate spare capacity.

2.6.1 Elasticity

An application's virtual resources in Cloud Computing can be scaled dynamically up, to maintain QoS and down, to reduce cost. This scalability is often refereed to as *elasticity*. NIST refers to elasticity as an integral aspect of Cloud Computing:

> "Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time." [104]

The rapid speed of resource provisioning is a key differentiating feature of the Cloud paradigm from other distributed systems. Virtual resources can be scaled in two directions:

- **Vertically (Scale Up)**: Refers to scaling the performance of an application by growing the internal resources of a running VM. This can include adding or removing CPU cores, CPU speed or fractional allocated physical CPU time, memory and data storage. Vertical elasticity must be supported by both the Hypervisor and guest operating system to enable the safe removal of virtual hardware devices.

- **Horizontally (Scale Out)**: Refers to scaling the performance of an application by increasing or decreasing the number of VM instances that supports its workload. Often horizontal elasticity is achieved through the use of hardware or software load balancer.

Vertical scalability is limited by the total number of resources available in single physical machine. On the other hand, Horizontal scalability is limited by the total number of physical machines available and by Amdahl's Law [7]. Amdahl's Law states that the maximum speed-up a program can gain by utilising multiple processors is limited by any sequential code that cannot be parallelised. In the context of Cloud Computing, this relates to the maximum number of requests an application can service given an infinite number of resources. This is often limited by the backend shared data storage that the VM resources access, rather than by the computation requirements of the application itself, which tend to have embarrassingly parallel properties by design in Cloud Computing.

2.6.2 Scheduling

Resources scheduling in Cloud Computing is an active area of research [92,155,219,286] with many academic institutions drawing on the research performed by the Grid Community on Market-Oriented scheduling policies [219,286]. Cloud resources scheduling based on a SLA is an NP-hard problem [155] with no method to provide an optimal schedule. The primary approaches to resource scheduling in Cloud Computing utilises the principles of resource load balancing [229]. In this scheduling approach, VM resource requests are considered to be "jobs" where the type of job specifies the VM hardware requirements, the size of the job specifies the amount of time the VM executes and job requests are scheduled to physical machines on a periodic basis. The following sections of the thesis highlight the scheduling algorithms currently in use in popular IaaS solutions.

2.6.2.1 OpenNebula

OpenNebula uses a match-making scheduling algorithm that assigns VM to known hosts using an rank based scheduling policy. The goal of this policy is to prioritise the use of resources that are most suitable for a given VM. The scheduler has a number of limitations. It is limited to scheduling a fixed subset of running VMs and dispatching a limited number of VMs in any given scheduling iteration. There is also a ceiling on the number of VMs that can be dispatch to a physical machine in any scheduling action.

The match-making algorithm used in OpenNebula filters host resource that cannot meet the CPU and memory resource requirements of a VM. A *RANK* expression specified by the end user is then evaluated against the remaining available hosts and the resources with the highest rank are used to allocate a VM. The available *RANK* policies are as follows:

- **Packing**: This policy minimises the number of physical machines in use, the heuristic packs as many VM as possible into a physical machine to reduce fragmentation, using those machines with the most running VMs first.
- **Striping**: This policy maximises the availability of resources in a host machine to a VM. VMs are spread across physical machines and those nodes that have the least running VMs are chosen first.
- **Load-aware**: This policy maximises resource availability similar to the previous policy but utilises physical machines with the least load. Thus physical machines with highest available free CPU time are selected first.

2.6.2.2 Haizea

Another example of a Cloud scheduler is Haizea [114], which can be used to replace the existing resource scheduler in OpenNebula. Haizea is an open source VM lease manager that provides a resource management model [22] for virtual Cloud infrastructures that combines batch execution of applications, such as scientific workflows, onto leased virtual resources [235]. Haizea supports the advanced reservation of resources, the queuing of best effort requests and the immediate provisioning of resources using the following lease types:

- **Advance Reservation**: In advances reservation a resource lease is provides between a specific start and stop date, useful when resources are needed at a specific time of day, for example when system logs need to be aggregated.
- **Best Effort**: Best effort leases are used for VMs that are not needed within a given time frame and are placed in a queue that is serviced on a first come first served basis. This lease type can be used with applications that support pre-emption and can be safely paused. Such applications are able to use resources quicker and this lease policy is well suited to batch jobs.
- **Immediate Lease**: Refers to the previsioning of resources instantly, typical of most Cloud scheduling systems.

In addition to scheduling functionality, Haizea can also be used to simulate resources for the testing of customised scheduling policies that can be plugged into its architecture. Schedule reports and graphs are also supported enabling algorithm analysis.

2.6.2.3 vSphere Distributed Resource Scheduler

The Distributed Resource Scheduler (DRS) [112,262] is a commercial VM scheduler provided by VMware as part of its vShere [268] IaaS offering. DRS provides mechanisms to distribute VMs over a a cluster of physical machines to maintain performance, scalability and availability. The following resources management functionalities are provided:

- **Initial Placement**: The creation of an initial schedule for the placement of VMs to physical resources.
- **Load Balancing**: Involves continuous workload monitoring and the live migration of VMs away from heavily used physical resources for the purpose of maintaining QoS.
- **Power Management**: Reductions in power usage through the migration of VM instances away from under utilised host resources enabling these hosts to be powered off or placed in standby mode.
- **Cluster Maintenance**: Non-disruptive maintenance of physical machines via live migration.

VMware have identified a number of challenges to Cloud based resources management [112]. These pertain to the heterogeneity of cluster hardware and the frequency of management operations. As IaaS providers add more machines over time to their infrastructure there can be several generations of hardware in coexistence. This can cause issues with the migration of VMs between hosts and the scheduling of workloads to hardware with differing levels of performance, which requires normalisation achievable via the execution of benchmarks that exhibit real world characteristics. In addition, as the number of users and the frequency of managements operations performed by a scheduler increase, so does the importances of maintaining low levels of scheduling latency.

2.6.3 Monitoring

Monitoring tools are essential in maintaining QoS and sustaining the performance of an application. Due to the the complexity of the Cloud Service Stack and the reliance on a range of infrastructure devices (network, storage, and computational resources) in the creation of a Cloud, there are a number of monitoring challenges to overcome [141]. Monitoring of distributed systems is not a new topic of research [129, 163] and in Cloud Computing many providers leverage existing monitoring technology.

The monitoring of Cloud resources, platforms and applications require different tools and techniques. Due to the dynamic nature of Cloud environments platform and application performance metrics are difficult to gather and monitor in the more advanced Cloud deployment models, such as hybrid Clouds. These difficulties are discussed in further detail in Section 6.4.1.2.

For the monitoring of Cloud infrastructure more traditional monitoring tools can be used. Ganglia [84] and Nagios [174] are two open source projects that provide tools that can be used to monitor IaaS Clouds. Both are used heavily in High Performance Computing (HPC) environments and have qualities that make them attractive to other distributed systems. In addition, both have taken on different stances in there definition of monitoring. Ganglia is more concerned with the gathering of metrics and tracks them over time, while Nagios has focused on alerting mechanisms for system administrators.

2.6.3.1 Ganglia

Ganglia [84, 163] is a scalable distributed monitoring systems that is based on a hierarchical design suited to the federations of clusters. It has been designed to achieve low overheads and high concurrency when gather monitoring metrics. Each machine monitored by Ganglia runs a daemon process named `gmond` that collects and sends metrics to a `gmetad` daemon that saves data to round-robin database files provided by the open source RDDTool [213], a high performance data logging and graphing system for the storing and visualisation of time series data. Multiple `gmetad` daemons can be associated with each other using point-to-point connections, enabling federated monitoring across geographic locations.

2.6.3.2 Nagios

Nagios [129, 174] provides a number of capabilities that make it suitable for monitoring applications, services, network protocols, operating systems, and infrastructure components. The Nagios architecture consists of the `Core` components and a API that allow the creation of plugins for the monitoring of custom applications and systems. The `Core` of Nagios provides out of the box monitoring for a number of network services such as SMTP, POP3, HTTP and NNTP in addition to host resources metrics such as processor load and disk usage. The architecture of Nagios enables the parallel checking of multiple services through the use of two addons:

- **Nagios Remote Plugin Executor (NRPE)**: An addon that enables the remote execution and polling of monitoring probes on host machines.

- **Nagios Service Check Acceptor (NSCA)**: An addon that provides the passive checking of remote metrics via an always on daemon process that sends data, which cannot be effectively gathered using polling techniques, asynchronously to a central repository. This approach is highly relevant to Cloud Computing due to its intrinsic distributed nature and resiliency to failure.

2.7 Resource Virtualization

In the previous sections of this chapter, virtual resource management was introduced as a critical concept used in the maintenance of QoS in Cloud Computing. In the following sections, the concept of Resource Virtualization is presented as a key enabler of virtual resource management in IaaS providers and is the foundation and primary building blocks of most Cloud applications. This background on Virtualization sets the scene for the research discussed in Chapter 5 on evaluating the performance of Cloud infrastructure.

Understanding Cloud architectures from the bottom up, starting with the technology that supports the provisioning of resources, both physical and virtual in Cloud infrastructures, is key to understanding the importance of QoS in Cloud Computing and how its implementation will differ, as it evolves, from that of Grid Computing. The current state of the art technology in Cloud Computing centres on the Virtualization of resources at the lowest level, a characteristic that distinguishes Clouds from Grids.

Virtualization refers to the creation of a virtual, as opposed to tangible, version of an object. This definition of Virtualization is broad and can be applied to many contexts. To narrow the applicable scope of Virtualization to distributed systems and Cloud Computing, Resource Virtualization refers to the isolation or combination of part or all of a computing device's hardware resource for different or shared purposes respectively. Resource Virtualization can be applied to three areas in distributed systems:

- **Network Virtualization**: Refers to a method of multiplexing the use of network devices while separating traffic, disguising the true complexity of the underlying network topology.

- **Storage Virtualization**: Involves the pooling of physical storage devices from multiple networked devices, presenting the appearance of a unified storage device, manageable from a central location.

- **Server Virtualization**: Provides isolated access to server resources while hiding the underlying implementation details of the hardware away from the End-User for the purpose of increasing utilisation and improving ease of management.

The main technology enabling Server Virtualization in Cloud Computing is the Hypervisor, often referred to as a Virtual Machine Manager or Monitor (VMM). A Hypervisor partitions a physical host server transparently via emulation or hardware-assisted Virtualization techniques, through a layer of abstraction. This provides a complete simulated hardware environment; know as a "*Virtual Machine* ", in which a guest operating system can execute in complete isolation. There are several benefits to utilising VMs. Hardware can be consolidated when several servers are underutilised and VMs can be provisioned as needed, endowing an organisation with reductions in the up-front cost of hardware purchases. Additionally, VMs can be migrated from one physical location to another, with ease and as the need arises, unlike traditional computing hardware that is difficult to move once deployed.

Academics can also benefit from utilising VMs. There are often limitations imposed on Grid users to what software they can use to develop scientific applications such as those that support computer based simulation experiments. There are no such limitations on the availability of software that can be installed into VM images as the End-User has `root` privileges.

Like many topics in distributed systems, research on Virtualization is not new. The first reference to the term "*Hypervisor*" was made in 1965 when IBM created software to enable memory resources of an IBM 360/65 to emulate that of an IBM 7080 [210]. The term originates from another: "*supervisor*", a process control system used to allow users to monitor and manage a number of processes. The Hypervisor process enabled the *fast* and performant switching between the two modes of memory operation. The performance of Hypervisors continue to be an active area of research [2, 118, 135, 139, 166, 175, 193, 204].

2.8 Hypervisor Classifications

To be able to classify a Hypervisor, the core properties that define one must be understood. Thus, a system is only capable of being virtualised by a Hypervisor, if the Hypervisor exhibits the following three properties [198]:

- **Equivalence**: Programs run through a Hypervisor should behave identically as compared to direct execution on equivalent hardware.
- **Resource Control**: A Hypervisor must have complete control over virtual resources provided to a VM.

- **Efficiency**: A statistically significant proportion of machine instructions must be executed without interference by the Hypervisor, pertaining to acceptable performance.

In addition, the seminal work by Goldberg et al. [93] classifies Hypervisors in to two categories, *Type 1*: Bare Metal (native) Figure 2.3(a) and *Type 2*: Hosted (embedded) Figure 2.3(b).

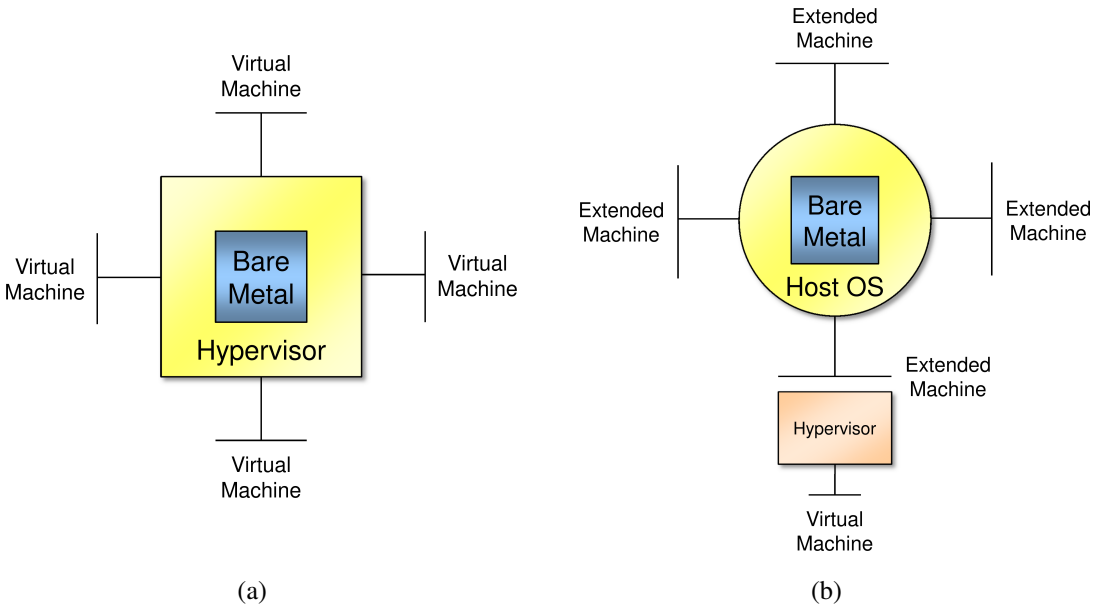


Figure 2.3. Hypervisor Classifications a) Type 1 b) Type 2.

2.8.1 Type 1 Hypervisor

In a *Type 1*, the Hypervisor runs on the host hardware with direct access and control of the underlying hardware resources. VMs execute at a second level above the Hypervisor. The Hypervisor must therefore perform the scheduling and resource allocation of physical hardware on behalf of running VMs. Typically, *Type 1* Hypervisors are used in datacenter environments within servers.

2.8.2 Type 2 Hypervisor

Type 2 differs from *Type 1*, in that the Hypervisor runs within a host operating system. The Hypervisor is given access to resources via the underlying host operating system's implementation, extending its functionality to enable Virtualization. In effect, VMs execute in a third layer of software above the hardware resources therefore incurring an

additional overhead. *Type 2* Hypervisors are usually found in desktop based Hypervisor varients.

2.9 Virtualization Techniques

There are five Virtualization techniques:

- Full Virtualization
- Hardware Assisted Virtualization (HVM)
- Partial Virtualization:
  - Paravirtualization
  - Hybrid Virtualization
  - Operating System-Level Virtualization

Full Virtualization involves simulating enough hardware to allow an unmodified guest operating system from a potentially different architecture to run in isolation, at a considerable performance penalty due to the overhead associated with emulating hardware at the transistor level. Hardware Assisted Virtualization utilises the additional hardware capabilities, currently in the form of Virtual Machine Extensions (VMX), within the host processor instruction set to accelerate and isolate context switching between processes running in different virtual machines. This increases the computational performance of a virtual machine, as instructions can be directly passed to the host processor without having to be interpreted and isolated, at the expense of limiting guest operating systems to using the same instruction set as the host machine. Complete Hardware Assisted Virtualization of all computer subsystems such as I/O and memory management, has yet to be implemented completely in any VMM.

Partial Virtualization involves the simulation of most but not all the underlying hardware of a host and supports resource sharing but does not completely guarantee isolated guest operating system instances. This basic approach is utilised in Paravirtualization, Hybrid Virtualization and Operating System-Level Virtualization. Paravirtualization simulates all or most hardware by providing software interfaces or API's which are similar to that of the underlying hardware of the host. These can be utilised to create virtual hardware device drivers for guest operating systems that achieve near native performance to that of the host. The downside of this approach is that the operating system must be modified to run on Paravirtualised VMMs. This is portrayed in Figure 2.4 where a privileged

Paravirtualised guest, known as Domain0 in the semantics of Xen [24], is responsible for turning virtual machines on and off, monitoring state and provisions access to the virtual device drivers and other resources within the Hypervisor, when a non-privileged Paravirtualised guest, known as DomainU, is brought online.

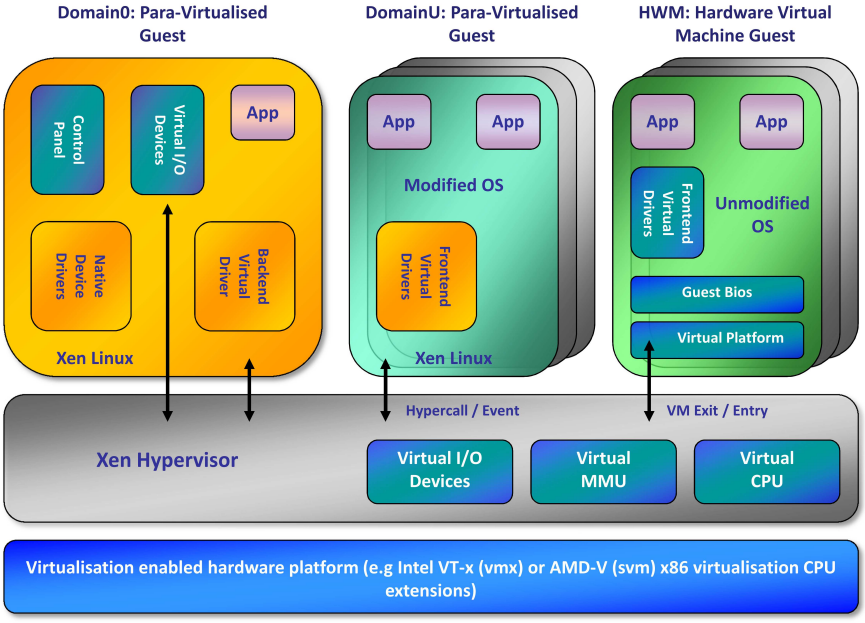


Figure 2.4. Xen Hypervisor

Hybrid Virtualization combines the principles of both Hardware Assisted Virtualization and Paravirtualization [175] to obtain near native performance from guest operating systems but with the disadvantages of both. Although these disadvantages prevent the consolidation of an organisation's current hardware they do provide an excellent foundation for the creation of new Cloud based systems, reducing the number of physical machines needed at peak demand and thus hardware running and setup costs. Most VMMs support multiple types of Virtualization so these disadvantages can be somewhat mitigated. Operating System-Level Virtualization is achieved through isolating multiple user space instances. A disadvantage of this Virtualization technique is that the guest operating system of the virtual machine must be the same as the host, but provides the benefit of the guest executing at native performance.

Not all Virtualization techniques are suitable for use in Cloud Computing infrastructures given the wide variety of applications that need to be supported. For example, legacy applications maybe best suited to the Full Virtualization technique if there is a dependency on a legacy operating system that runs on a different computer architecture. On the other hand, applications that have high performance requirements, stipulated as part of QoS in

a given SLA, would be better suited to Hypervisors that support the Hybrid Virtualization technique. Additionally, applications that only support operating systems that are not modifiable by third parties, such as Windows, would be best placed on Hypervisors that support the Hardware Assisted Virtualization technique.

2.9.1 Live Migration

Migration refers to the movement of a VM from one host to another. The migration of VMs between hosts is a core functionality of an IaaS provider and enables the optimization of VM workload allocation to physical resources. Without this functionality workload consolidation would not be possible and Cloud resources would be underutilised. The migration of a VM from one host to another incurs a downtime penalty when the VM is in transition.

Live migration refers to moving a running instance of a VM from one host to another over a network while still servicing requests and incurs minimal downtime. Live migrations brings a set of challenges with regards to maintaining the QoS of the running VM in migration and other VMs resident in the source and destination host [16]. Live migration induces a load on CPU, network and memory bandwidth, when the host machine's Hypervisor copies the state of the running VM instance. Live migration of a VM occurs in three phases [46]:

- **Pre-Copy (Push)**: The VM continues to run on the source host machine while memory pages of the VM that are least used are pushed over the network to a new destination host. Pages modified on the source during this phase are re-transmitted.
- **Stop-and-copy**: At a point where either all pages are copied over to the destination host or the rate at which pages are being dirtied by the VM on the source exceeds QoS restraints or network bandwidth, the source VM is stopped and any remaining pages are copied over. At this point a new VM is started on the destination machine with the source's state.
- **Post-copy (Pull)**: This phase maybe necessary if time constraints are placed on the live migration process, leaving pages to be sent on the source machine. Pages of memory are *pulled* across the network when a page fault occurs on the destination VM, when a VM requires access to memory pages that have not yet been transmitted.

Live migration is not suitable for all applications as there is a trade off point where the speed at which pages of memory are dirtied can exceed the available bandwidth to transfer

them over the network. In addition, live migration can adversely effect other VMs due to resource contention between the multi-tenant VM instances and the Hypervisor's migration process running on the same host machine. The performance of Live Migration is an active area of research [16, 18, 68, 116, 119, 121, 136, 156, 237, 242, 282]. More so, the live migration of VMs across Wide Area Networks, highly applicable to Cloud Computing, is still to be fully realised due to issues of interoperability between Hypervisors and is an area of on going research [31, 159, 274].

2.10 Virtual Machine Images

A VM disk image is a representation of the contents and structure of a storage medium or device, such as a hard disk, USB flash drive or optical discs. A VMM access data via these images to enable multiplexing of data resources. The contents of the image contain data representing a sector-by-sector copy of a storage device at the guest level. There are many types of disk image format, of which some provide additional features like excluding unused sectors, reducing the physical disk space required to store the image and encryption for enhanced security. In this section of the thesis, VM disk images are discussed in the context of Cloud Computing.

2.10.1 Storage Backends

In Cloud Computing there are different ways to access and store VM images:

- Local storage provided by either:
  - File Backed (File System Managed)
  - Raw Block Device (No File System Management)
- Remote storage either via:
  - NAS: Network Attached Storage (File System Managed)
  - DFS: Distributed File System (File System Managed)
  - SAN: Storage Area Network (No File System Management)

2.10.1.1 Local VM Storage

Local storage refers to storing and accessing the data representation a VM on the same physical host as to which a VM is executed on. There are two ways this can be achieved:

File Backed or virtual mode storage provides a VMM with a disk file that sits on top of the local host file system, obscuring the real hardware characteristics of a physical block device. In this mode, file level locking provides data protection when multiplexing a storage resource. In addition, this mode offers portability across storage hardware via a layer of abstraction above the host machine. A typical solution to File Backed storage in a Cloud would require the staging in and out of a VM image to and from a physical host machine via a centralised image repository and would make use of a file transfer protocol.

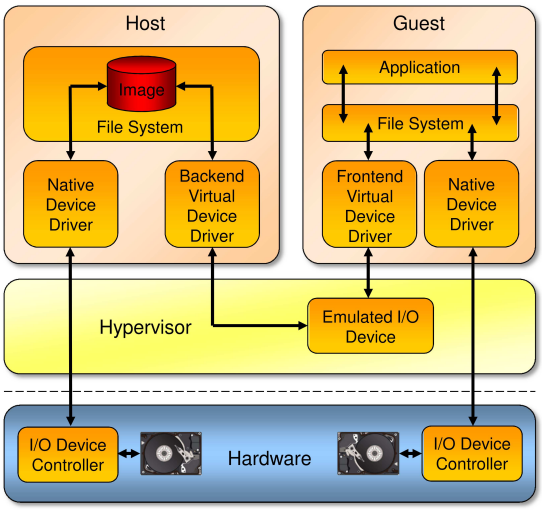


Figure 2.5. Overhead of Using Ffile Backed Storage (Left Path) Against Raw Block Storage (Right Path)

In *Raw Block Device* or physical mode storage, the VMM bypasses the host file system and the I/O Virtualization layer as shown in Figure 2.5. All I/O commands are passed directly to a physical raw block device on the host machine. The physical characteristics of the underlying storage hardware are exposed to the guest operating system and provides no data protection. Common solutions use Logical Volume Management (LVM) to enable the dynamic deallocation and allocation of space to a virtual machine after its creation. With this, a single physical disk can be fragmented up into virtual volumes and assigned to a specific VM.

2.10.1.2 Remote VM Storage

Remote storage refers to storing a VM image in a physical location separate from the computational resources of the guest to which it belongs. There are three ways remote storage can be implemented as illustrated in Figure 2.6:

Network Attached Storage refers to remote storage made available through a TCP/IP

network using specific protocols such as the NFS and Common Internet File System (CIFS), which provide interaction with data at the file level. NAS is similar in principle to local file backed storage where a VM image is stored on top of a file system managed by the NAS device. Where NAS differs is in providing access to files from a remote location from a single physical machine.

Distributed File System can be conceptually thought of as an extension to NAS where by discrete storage resources on multiple machines can be combined and shared amongst many VMs for improved scalability. In addition, a DFS may provide features for providing transparent replication and fault tolerance of data for high availability purposes above and beyond Redundant Array of Independent Disks (RAID) on a single machine.

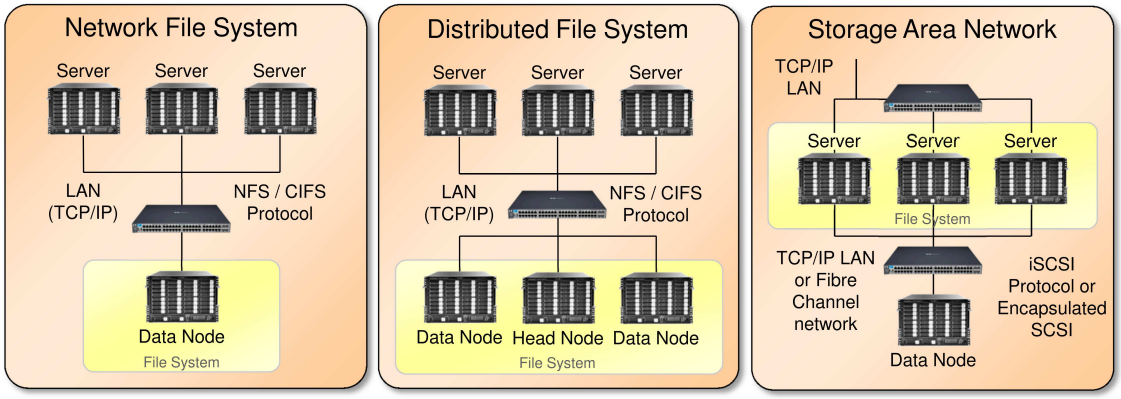


Figure 2.6. Topology Differences Between NAS, SAN and DFS

Storage Area Networks address data at the disk block level and transfers raw disk blocks over a network. This approach is similar to local raw block device storage in that no underlying file system is provided, the file system is managed by the guest OS. Traditionally a SAN would use the Small Computer System Interface (SCSI) carried over a fibre channel connect enabling shared block level storage up to distances of 10km. SCSI is a set of standards defining commands protocols and electrical and optical interfaces for connecting and transferring data between machines and peripheral devices, most commonly used for disk drives. More recently with the advent of low cost high speed Ethernet adapters and as administrative domains have increased in size, the Internet Small Computer System Interface (iSCSI) has risen in prevalence. The iSCSI protocol wraps SCSI commands in IP packets for transportation over long distances including Wide Area Networks (WAN) such as the internet.

2.10.1.3 Selecting a Storage Backend for the Cloud

There are both advantages and disadvantages to using local or remote storage backed by either a file system or not backed by a file system in a Cloud environment. No one solution is appropriate for all Cloud applications.

Local storage is not limited by network connectivity and has a minimal computational cost due to not having an associated network message processing and latency overhead but does not scale with increasing space requirements and is difficult to manage administratively across a Cloud infrastructure. In addition, network storage can have an effect on other application level traffic if a dedicated network is not used.

Using a file backed image on top of a file system improves portability of VMs between hosts and the management of VMs across a Cloud infrastructure but has an associated performance overhead where by requests for data have to traverse both the structure of the guest's file system and the host's file system before a block of data can be accessed at the hardware level. There can also be Virtualization and image format overheads, which will be discussed later in this chapter in Section 5.3.3. In contrast presenting a raw block device to a VM enables native I/O performance within a guest but in the context of local storage can complicate or prevent migration of VMs between physical hosts.

Although there are drawbacks to using local storage with file backed images, there are advantages with regards to interoperability in more advanced Cloud environments, such as with a federation of Cloud providers spanning geographically disperse data centres. Images can be easily propagated to heterogeneous computational resources with little effort in an ad-hock fashion using conventional file transfer protocols such as the File Transfer Protocol (FTP) and Secure Copy (SCP). Support for this approach to image management is available in all VIMs and is discussed in detail in Section 5.2.

2.10.2 Virtual Appliances, Packaging and Distribution

VM images can be packaged together to form a virtual appliance for ease of distribution. A virtual appliance is comprised of many different types of VMs that contain specific elements of an application's software stack, referred to as "virtual systems". An open standard used by many Cloud infrastructure managers to package virtual appliances is the Open Virtualization Format (OVF) [192] standard.

An OVF package, often confused as a VM image format itself, specifies the hardware requirements and configuration of a image at creation time and can contain or reference other files such as VM images. The description itself is a human readable XML document that contains metadata that includes naming and hardware requirements of one or more

VMs.

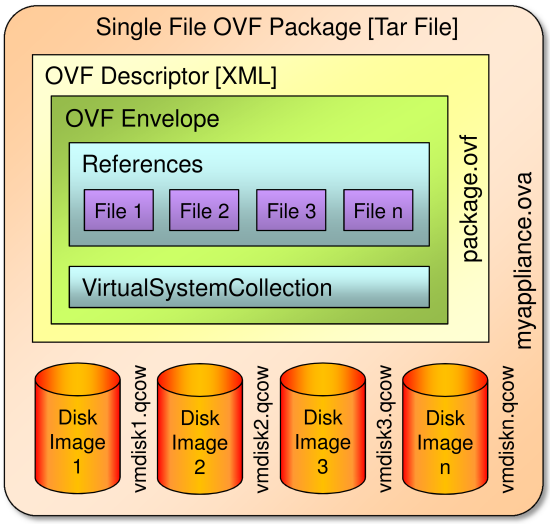


Figure 2.7. An OVF Packaged Virtual Appliance

The OVF standard has found wide spread support in industry and in the open source community by enabling interoperability between vendors as the standard is not tied to a specific VMM. A potential drawback to using the standard is that some specific virtual hardware features can be lost in the semantic abstraction when catering for the lowest common denominator of feature set to enable portability across all VMMs.

2.10.3 Image Formats

There are several VM image formats that are available for use in Clouds with varying degrees of support at the Hypervisor level. In this subsection, the features and the Hypervisor support of all disk image formats currently in use in Clouds, are explained and presented in detail as a taxonomy.

In Table 2.1 a comprehensive list of image formats, the features supported by each and the VMM of origination where applicable, are presented:

Table 2.1. Comparison of Image Format Features

Image Format Name	**Features**	**Hypervisor of Origin**
QCow2 [257]	Dynamic Allocation, Preallocation, Cluster Size Selection, Encryption, Compression, Snapshots	QEMU
VMDK [259]	Dynamic Allocation, Preallocation, Snapshots, Split Disks	VMware
VHD [256]	Dynamic Allocation, Preallocation, Snapshots	Microsoft Virtual PC
VDI [255]	Dynamic Allocation, Preallocation, Snapshots	VirtualBox
Raw	Dynamic Allocation, Preallocation	N/A
AMI [8]	Dynamic Allocation, Preallocation	Amazon EC2

Preallocation refers to virtual disk storage that is allocated in its entirety upon creation. The opposite is true of *Dynamic Allocation* where virtual disk storage grows on-demand as new disk space is required. Preallocation is preferable if performance is of primary concern as there is an overhead associated with growing an image.

The *Snapshots* feature refers to using a copy on write mechanism. This is analogous to differencing, enabling changes made to a disk to be undone. All changes to an image are contained within a separate file or child image and can be merged to the parent file if needed at a later date. Copy on write operations can also be used to share a base image between running VMs instances, where concurrent updates are isolated and stored in a snapshot file, reducing data storage requirements and the time to create a new VM instance as the base image does not need to be copied to be cloned.

In addition to the previously mentioned features, QCow2 supports Cluster Size Selection, which enables fine tuning of performance in contrast to the image size, where a smaller cluster size reduces the overall image size and a larger cluster size provides enhanced performance. Optional full image level pass phrase AES *Encryption* for enhanced security and zlib *Compression* for smaller image sizes, are also supported by this format.

The VMDK image format uniquely provides *Split Disks* that enables support for older files systems such as the File Allocation Table (FAT) file system, where there is a file size

limit of 2GB. Further more, split disks reduce the space needed to consolidate snapshots.

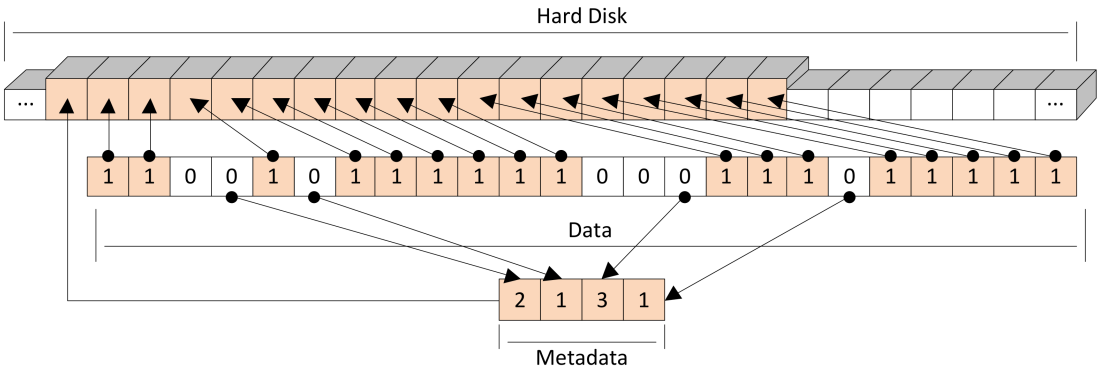


Figure 2.8. Illistration of Sparse File Support in a File System

The raw image format differs from the others discussed in that it does not contain metadata or a format header and only supports dynamic allocation on file systems that provide ”holes” or sparse files, where blocks of data with a zero value do not point to a physical location but are instead recorded in the metadata of the file system. This feature is illustrated in Figure 2.8 and is available in the Fourth Extended File System (Ext4).

In Table 2.2 a taxonomy of image support, in a number of Hypervisors, is presented. The scope of the taxonomy is limited to Hypervisors that support `x86` and `x86_64` hardware, all guest operating systems and have been created for use in server environments, i.e., VMMs that operate through Virtualization, Paravirtualization or Hardware Assisted Virtualization. Container based VMMs that use Operating System-level Virtualization are of limited usefulness and desktop Hypervisor solutions are not applicable in Cloud environments. A VMM is also considered to support a specific image format if the image can be run natively without requiring conversion before use.

Table 2.2. Comparison of Image Format Support in VMMs

Virtual Machine Manager	**Image Format Support**	**Comments**
KVM	QCow2, VMDK, VHD, VDI, Raw	Image support via QEMU device emulation.
Xen	QCow2, VMDK, VHD, Raw,	Raw and VHD image support via blktap2 device driver. QCow2, VMDK, VHD image support via integrated QEMU device emulation and file device driver backend.
VirtualBox	VDI, VMDK, VHD	Newly created guests via VDI format only.
VMware ESX Server	VMDK	Version 3 limited to 2TByte volumes
HyperV Server	VHD	Limited to 2TByte volumes

KVM provides the most comprehensive support for image formats with its continued upstream integration with QEMU. Xen follows closely behind as it advances toward KVM to provide user space HVM guest support via QEMU Paravirtualized device emulation. Currently Xen makes use of an older version of QEMU with reduce image format support. In addition, Xen supports its own implementations of the raw image format and VHD for Paravirtualized DomainU based guests. The Xen implementation of the raw image format is a based on an unpartitioned raw loop-back image that can be used in conjunction with the kernel space `blktap` device driver. Unfortunately this driver is poorly maintained and the lack of partitioning in the raw image causes difficulties when migrating to other Hypervisors.

An evaluation on the performance of each of the image formats discussed within the context of all the available open source Hypervisors at the time of writing, is provided in Section 5.4.4.1.

2.10.3.1 Image Maintenance and Management Issues

VM images need to me managed and maintained to be of use in a Cloud Computing environment. This includes such tasks as cloning, conversion, encryption and compression. One particular issue that is yet to be resolved regarding image maintenance is the problem of reclaiming deleted space from dynamically allocated image formats. When data is deleted from an image it is only dereferenced at the level of the guest file system. This leave the deleted data present in the underlying host file system. One possible solution to this problem could be to implement a similar command to `TRIM` [83], at the Hypervisor block I/O device level, to maintain compact image sizes over extended periods of use. The `TRIM` command is used by operating systems backed by Solid State Disk (SSD) drives, where by deleted blocks of data are physically zeroed upon deletion for the purpose of improving performance when data is written in the future. In Virtualization the same technique could be used to zero deleted block of data within an image. An attempt at maintaining sparsity has been considered at the kernel and file system level by [211].

Another issue regarding management, relates to the conversion of images between Hypervisors. Although it is easy to convert between image formats it is more difficult to abstract away internal changes made at the guest Operating System level. To obtain higher performance the Operating System of a VM can be installed with and configured to use Paravirtualized device drivers, which are often Hypervisor specific and incompatible from one Hypervisor to another. This can prevent a virtual machine from booting and is discussed further as part of contextualization in Chapter 6.

2.11 Contemporary Hypervisors

There are several VMMs that are widely adopted by Cloud Computing architectural deployments. These come from a variety of sources including the open source community and commercial vendors.

2.11.1 Open Source

This section discusses three open source Hypervisor variants. All the open source Cloud architectures discussed in Section 2.4 support at least the following two VMMs.

2.11.1.1 KVM

The Kernel Based Virtual Machine (KVM) [153,205] is a Type 2 Hypervisor that supports Hardware Assisted Virtualization and Hybrid Virtualization techniques. KVM primarily supports `x86` and `x86_64` processor architectures. Recent versions have however been ported successfully to PowerPC and IA-64 (Itanium) architectures. Additionally, the development of KVM is supported by Red Hat.

The approach taken by KVM takes a standard Linux kernel and turns it into a Hypervisor by simply loading a kernel module. The architecture of KVM is split between a Linux kernel module and user space program. KVM itself does not perform emulation of devices but instead provides access to hardware Virtualization features available in various processors through the `/dev/kvm` device interface. This device is presented to user space programs enabling isolated address spaces between VMs. A VM's virtual CPU is implemented using regular Linux processes and threads that are scheduled by the Linux scheduler which enables KVM to leverage the many resource management features within the Linux kernel.

Device emulation is performed by a user space program that makes use of the `/dev/kvm` interface to feed a VM with simulated I/O and provide a virtual graphical display on the host. QEMU [200] since version 0.10.1 has made use of this device interface. KVM with QEMU supports a number of guest operating systems variants based on Linux and Windows.

As KVM makes extensive use of the Linux kernel to manage physical resources, it has thus inherited its performance and scalability. VMs can make use of up to 16 virtual CPUs and 256GB of RAM on host machines with up to 256 logical CPU cores and over 1TB of RAM. In addition, consolidation rations of more than 600 VMs running on a single physical host have been recorded [205].

Advanced scheduling and resource control mechanisms within the Linux kernel can be leveraged to set resource priorities of VMs including CPU, memory, network and disk I/O to maintain QoS. Recent extensions to the Linux kernel and its Completely Fair Scheduler (CFS), have seen the implementation of `CGroups` or control groups to manage resources at a process level. `CGroups` enable the sharing of resources beyond the weighting of resources typically found in other Hypervisor implantations. This enables resource minimums as well as maximums to be assigned to a VM, guaranteeing resource allocations.

2.11.1.2 XEN

XEN [24] makes use of the Hybrid Virtualization technique, is utilised in Amazon's EC2 IaaS and was originally developed by the University of Cambridge but is currently supported by Citrix Systems. It supports `x86`, `x86_64` and ARM based computing architectures and a number of guest operating system variants based on Windows and Linux.

XEN is a Type 1 Hypervisor with an architecture that makes use of a micro-kernel, which is booted before a Domain0 privileged host operating system illustrated in Figure 2.4. A micro-kernel is the minimum amount of software needed to implement the functionality of an operating system. The Hypervisor executes in a higher privileged state than the Domain0 or DomainU guests. The XEN micro-kernel manages memory and CPU scheduling of VMs, while the Domain0 privileged guest, a modified version of Linux, manages access to and Virtualization of devices.

XEN is highly scalable and has been tested on host machines with greater than 255 physical CPUs. VMs can be allocated up to 128 virtual CPUs and 1TB of RAM [277]. KVM and XEN are compared in detail as part of Section 5.3.2.

2.11.1.3 VirtualBox

VirtualBox [258] is a Type 2 desktop Hypervisor utilising Hybrid Virtualization that can be run in "*headless*" mode on server hardware. It comes in two versions, an open source variant and a closed source free version that makes use of an extension pack for support of Microsoft's Remote Desktop Protocol (RDP), USB 2.0 devices and Intel network cards, which do not fit with the GPL license model used in the open source version. Widespread adoption of this Hypervisor in IaaS architectures and providers is not yet a reality, due to the performance overheads of hosted Type 2 Hypervisors and is thus currently relegated to desktop Virtualization use cases.

2.11.2 Commercial Variants

The commercial alternatives to XEN and KVM in Cloud Computing infrastructures are as follows:

2.11.2.1 VMware

VMware [260] provide a number of Hypervisor products with different applications and deployment scenarios in mind:

VMware ESX [261] is a successful commercial VMM that is used in enterprise datacenter environments. It is a Type 1 Hypervisor that provides direct access to physical resources on behalf of VM guests. It provides Full and Hybrid Virtualization techniques through a Linux based micro-kernel. The Hypervisor provides interfaces to: hardware, VMs and a management console. It supports most Linux and Widows variants on `x86` and `x86-64` architectures but does not support RISC based operating systems such as PowerPC. Guest systems are supported with virtual resources of up to 1TB of RAM and 32 virtual CPUs. Host machines are constrained to 2TB of RAM and 160 CPUs with a maximum number of 512 VMs per host.

VMware Workstation [265] and Server [263] are Type 2 Hypervisors that are orientated towards desktop and datacenter environments respectively. VMware Server has fewer features than ESX but can be deployed onto existing Windows host operating systems. VMware Workstation enables desktop Virtualization and has support for 3D hardware accelerated applications that make use of both Direct3D and OpenGL graphics. Currently, GPU Virtualisation techniques exhibit poor performance but as virtualised GPU devices improve, further cost savings will be released in Cloud Gaming, outlined in Section 2.5.3, enabling a greater number of games to be supported.

VMware Tools, plays a critical role in the Paravirtualization of guests on all VMware Hypervisors. It is a package of drivers that improve the performance of a guest operating system and adds additional functionality like clipboard sharing and system clock time synchronisation. Paravirtualization is achieved through an open standard, the Virtual Machine Interface (VMI), a communication channel to the Hypervisor from within a guest.

2.11.2.2 Hyper-V

Microsoft's Hyper-V [122] is a Type 1 Hypervisor that uses Hardware Assisted Virtualization that is often confused as Type 2 due in part to it being bundled with Windows Server 2008. In fact, the Hypervisor loads before the host operating system used to manage VMs and runs directly on the physical hardware. Hyper-V requires hardware based on the `x86_64` architecture and is limited to 384 guest VMs per system, with each guest able to support a maximum of 4 virtual CPUs. Hyper-V has support for all versions of Windows from the year 2000 but is limited to enterprise Linux distributions such as those supported by Red Hat and SUSE.

2.11.3 Operating System-Level Variants

Operating System-Level container based Hypervisors are classified as Type 1 due to the integration of the Hypervisor resource management code into the host operating system's kernel. These variants are not well suited to IaaS roles as they are limited to a single operating system type and lack support for VM image formats but are popular with Virtual Private Server (VPS) providers that host websites for End-Users. Web site supporting software stacks, such as those that use Apache, Mysql and PHP, are not constrained by the single operating system limitation and tend to be highly portable. The value added by VPS providers in provisioning VMs compared to the provisioning of shared web hosting, is the advanced configurability and customisation it enables the End-User to make to the supporting software stack.

2.11.3.1 OpenVZ

OpenVZ [190] is an open source container based Virtualization for Linux, is supported by *Parallels, Inc.* and is the basis of their commercial Hypervisor offering: Virtuozzo Containers. The Hypervisor can be used to create multiple isolated Linux containers that reside on a single physical server. Each container has access to its own files, users and groups, process tree and virtual network device. Applications running within an OpenVZ container are limited, if configuring of kernel-level features are required, due to the use of a shared kernel between all instances. In addition, only VPN technologies that run in user space are supported, excluding the use of enterprise grade Internet Protocol Security (IPsec) tunnels that run at the kernel level.

2.11.3.2 VServer

Linux VServer [267] is another open source Linux container based Virtualization solution similiar to OpenVZ but has a number of additional disadvantages. These include a lack of support for container migration between hosts, no support for the allocation and sharing of disk I/O between running instances and parts of the filesystem are left unvirtualised, reducing the effective isolation between guests that in turn has an impact of security.

2.12 Summary

In this chapter of the thesis, the topic of Cloud Computing, its heritage and deployment models were introduced. A classification of Cloud services was presented that categorises Clouds into three models: IaaS, PaaS, IaaS. Additionally, a number of emergent Cloud

types were discussed. Several open source architectures, usable in the creation of a Cloud provider, were examined in addition to a number of commercial Cloud offerings, providing an overview of the technological landscape that forms the Cloud ecosystem. The topic of resource management in Cloud Computing and the concept of elasticity was discussed. Furthermore, resource scheduling in the context of Clouds and Virtualization was investigated. In addition, a number of scheduling implementations were examined, the relevance of monitoring tools discussed and applicability of two monitoring system implementations were presented.

Additionally, this chapter of the thesis introduced the topic of Resource Virtualization and its applicability to Cloud Computing as the basic resource building block of any IaaS provider. The properties that define and a formal classifications of Hypervisors were presented, in addition to a variety of Virtualization techniques used to enable the provisioning of VM resources. The usefulness of these techniques in Clouds were discussed in light of the effect each technique has on performance and whether a technique is particularly suited to a given Cloud application. Additionally, the topic of Live Migration and its application in maintaining QoS was presented. Finally, a number of contemporary Hypervisor variants in active use were discussed.

Chapter 3

Cloud Application Composition

3.1 Introduction

The previous chapter described virtual resources as the foundation on which a Cloud application is built, relevant to the lowest level of the Cloud Service Stack, IaaS. Building applications in Cloud Computing requires the collaboration of components across all levels and thus encompasses a number of disciplines from within software engineering. This chapter outlines the concepts and technology that enable the composition of technology and components into Cloud born applications. The background on composing Cloud applications presented in this chapter, sets the scene for the research performed in Chapter 6 on the Contextualization of components for the purpose of orchestration.

The outline of this chapter is as follows: Section 3.2 introduces Service Oriented Architectures, including the concept of service orientation, web service technology and service orchestration in relation to Clouds. Section 3.3 discusses the emergence of Cloud Engineering including the challenges it must face and the requirements needed to apply it successfully. Additionally, the relevance of the Web Scale Computing concept and a number of design patterns are discussed. Finally, Section 3.4 discusses the migration of applications to the Cloud, introducing simulation as a mechanism to validate the benefits to QoS, the possible pros and cons of using Cloud services within an application's software stack and best practice to apply when the decision has been made to make use of a Cloud and application development is about to start.

3.2 Service Oriented Architectures

In the paradigm of Cloud Computing, the concept of Service Oriented Architecture (SOA) plays a critical role in the development and integration of services across the entire Cloud stack, providing support to an application and enabling the fulfilment of its non-functional and functional requirements.

SOA provide a number of principles and methodologies for the design and development of interoperable software services that can be leveraged in a Cloud, specifically in the PaaS and SaaS layers. In SOA, a service is a well defined set of functionalities that is combined together as a software component for the purpose of reusability. In Cloud Computing the design principles of SOA enable the consumption of SOA services within a Cloud application using dissimilar programming languages. The interface of a service is often defined using XML, enabling the integration of widely disparate service implementations and platform technologies.

SOA relies on the principle of loose coupling, where service functionality is independent with minimal external dependencies and no calls to other services are embedded in a service's source code. Instead a service uses a well defined protocol to communicate, which describe how messages are passed and parsed. Service functionality is thus only made available over a network.

3.2.1 Service Orientation

In SOA, the design principles of Service Orientation govern application development. The principles have been outlined as follows [245]

- **Standardized Service Contract**: Services adhere to a communications agreement defined collectively by service description documents.
- **Service Loose Coupling**: Services maintain relationships that minimize dependencies, requiring awareness of service functionality only.
- **Service abstraction**: Descriptions hide service implementation and logic from the outside world.
- **Service Reusability**: Logic is divided into services with the aim of promoting reuse.
- **Service Autonomy**: Services are independent of the logic encapsulated.

- **Service Statelessness**: Services minimize resource usage by deferring the management of state where possible.
- **Service Discoverability**: Services are accompanied with metadata used to discover and interpret functionality.
- **Service Composability**: Services must be composable with one another regardless of the size and complexity of the composition.

These principles are highly applicable to the Cloud Service Stack, which currently supports limited interoperability and programmatic discovery of Cloud services. In addition, the application of these principles results in the creation of software units that are partitioned by capability and designed to solve a specific problem. In the context of Cloud Computing, Internet based protocols can be leveraged to make these functional building blocks accessible as *Web Services*.

3.2.2 Web Services

A Web Service at the highest conceptual level, is a method for enabling communication between devices over the World Wide Web (WWW). W3C [251] has defined a Web Service as a ”software system designed to support interoperable machine-to-machine interactions over a network” [271]. A Web Service’s interface is described using a machine readable format or markup language that is consumed by a client to enable the calling of a Web Services functionality. These descriptions can be stored in a Universal Description Discovery and Integration (UDDI) repository for the purpose of listing a web service on the Internet.

The developers of Web Services make use of standards to enable greater interoperability and avoid vendor lock-in. Two of the most popular service based protocols for communication are SOAP [269] and REST [212].

3.2.2.1 Simple Object Access Protocol (SOAP)

SOAP provides a specification for the structured exchange of information using XML. SOAP services are expressed as machine readable description of service operations which can be written using the Web Service Description Language (WSDL) [275] although this is not a requirement it is necessary for the automated generation of client and server code.

SOAP can make use of a number of different transportation methods such as TCP, UDP, SMTP and HTTP. The HTTP web based protocol has become a de facto standard

for transportation, as the protocol is able to traverse firewalls and leverage the security and identification features already present in the protocol.

There are a number of criticisms of SOAP. Namely the verbose nature of the XML messages used cause performance related issues when small messages sizes are sent. In addition, the reliance on the client-server architecture of HTTP as a transportation protocol prevents the bi-directional communication of messages. Clients are only able to call a service thus limiting a developer to the inefficient polling of a service to detect changes in state.

3.2.2.2 Representational state transfer (REST)

Representational state transfer (REST), takes an alternative approach to Web Service communication. REST does not mandate the use of XML, SOAP or WSDL but instead constrains the use of a protocol, for example in HTTP, by standardising on a set of well known operations:

Table 3.1. HTTP Based Rest Methods

Resource Type	**`GET`**	**`PUT`**	**`POST`**	**`DELETE`**
Collection	**List** the resources of a collection.	**Replace** all resources in a collection.	**Create** a new resource in the collection.	**Delete** the entire resource collection.
Element	**Retrieve** a representation of a resource element.	**Replace** a representation of a resource element or if it doesn't exist create it.	**Create** a new resource representation of an element.	**Delete** the resource representation of an element.

The focus also differs, interactions are made with stateless resources rather than concentrating on the standardised transfer of messages, as in SOAP or on the calling of methods, as in the case of RPC implementations. Requests and responses in REST are thus built around the transfer of resource state representations. A client sends a request to

move a service's resource into a new state and the service returns a representation of the resource's state to the client. The client while operating on this representation is considered to be in transition. After the client completes its transition, a new representation of state is returned to service.

3.2.2.3 Markup Languages

Markup languages play a role in the annotation of Web Services and enable the automated composition and implementation of service designs. Automation tools can either generate service descriptions from existing code or consume them to create code stubs. WSDL is an example of one such markup language specification used in SOAP. WSDL describes a service as a collection of network endpoints which are associated with a binding. Endpoints specify invokable methods and likewise a binding represents a reusable message description.

3.2.3 Service Orchestration

Service orchestration refers to the automated deployment, management and coordination of services. The orchestration of services is based largely on Control Theory [70] where the behaviour of a system is managed via the manipulation of control variables that are used to apply desired effects on a system. In addition, it helps define policies for the automated workflow management and resource provisioning of services. Service orchestration is highly applicable to SOA, Virtulaization and Cloud Computing where it describes the centralised management of resources pools, including billing/metering and the composition of services to solve specific problems using reusable system components.

In the context of Cloud Computing, service orchestration is the composition of architectures, tools and processes to deliver a defined service. Software and hardware components are combined together alongside the automation of IT workflows. This enables the delivery of Cloud services at appropriate scales of economic feasibility while maintaining applicability to both the domains of a business and technical processes.

3.3 Cloud Engineering

Cloud Engineering refers to the application of engineering principles in the context of Cloud Computing for the purpose of enabling a systematic approach to the creation of commercially viable and standardised Cloud services. Cloud Engineering draws on many

different disciplines such as system, software, web, performance and security engineering. The commodity based capabilities of Cloud services bring a number of inherent challenges that must be overcome to realise the full potential of the paradigm. In addition, the complexity of Cloud Systems drive the need for Cloud Engineering principles.

3.3.1 Challenges

There are a number of challenges facing Cloud Engineering. The relative youth of the Cloud paradigm and the revenue protectionism of Cloud providers have an adverse effect on standardisation of Cloud technologies. Cloud Engineering must accommodate for application and data portability as well as dealing with issues of scalability in and across Cloud providers.

Another challenge facing a Cloud engineer is the monitoring and management of applications. The holistic nature of Cloud applications across the entire service stack pose a number of issues for traditional on premise enterprise applications. With the inclusion of virtual resources, capacity management at the Hypervisor level must be monitored to maintain performance. Traditional monitoring systems concerned with physical systems that remain static are not well suited to the dynamically changing environment of the Cloud.

The scale enabled by Cloud infrastructure bring another challenge related to the profiling and testing of Cloud applications. Traditional applications are bound by predictable and measurable usage patterns. The scale experienced by Cloud applications are not. Traffic spikes are seemingly unpredictable and unlimited in nature.

The externalisation of resources from an organisation also create a number of challenges when engineering secure Cloud applications. Not only must the frontend of the application be secured from potential unauthorised access but so to must the backend resources, which are left potentially accessible by third parties sharing the same resources and the Cloud provider.

Finally, multitenancy makes the profiling of individual client performance indicators difficult as applications competing for physical resources have an adverse effect on one another. Applications developers when testing must thus be able to distinguish between not only the overall health of their services but the potential impact other services running in the same infrastructure could have.

3.3.2 Requirements

These challenges thus bring a number of interesting requirements for the systematic engineering of a Cloud service including Standardisation, Scalability, Security and Manageability.

Standardisation enables a Cloud engineer to create applications and services that require minimal redevelopment effort to be deployed onto a range of Cloud infrastructures and platforms. There are a number of standards organisations and associated working groups developing standards for Cloud Computing:

- **Cloud Security Alliance** [48]: Formed to promote a series of best practices for assuring security in Cloud Computing. It has a number of objectives including the promotion of understanding, researching best practices and launching awareness campaigns with the goal of creating consensus on measures to ensure Cloud security.

- **Open Cloud Consortium (OCC)** [182]: The goal of OCC is to develop standards for Cloud Computing and create frameworks for interoperability between Clouds. A number of working groups are devoted to various aspects of Cloud Computing including virtual networks and open science data Clouds.

- **Cloud Computing Interoperability Forum (CCIF)** [39]: CCIF is a forum for discussing the creation of the Cloud ecosystem promoting organization working together through the use of Cloud Computing technologies. The primary focus of the forum is to create a framework for enabling Clouds platforms to exchange information in a unified way.

- **Open Grid Forum (OGF)** [183]: OGF is a community that focuses on the adoption and evolution of distributed systems technology. This includes a number of topics such as HPC and SOA as well as Cloud Computing.

- **The Object Management Group (OMG)** [184]: OMG is an international group with a focus on developing enterprise standards for software intergeneration and encompasses a wide range of industries including governments and healthcare. The group creates modelling standards for software along side other business processes.

- **Distributed Management Task Force (DMTF)** [61]: The DMTF is focusing primarily on IaaS and creating a number of standards to enabling flexible, scalable and high performance infrastrucutre. This group contributed to the development of the Open Virtualization Format (OVF) [192] discussed later in Chapter 5.

- **Storage Networking Industry Association (SNIA)** [233]: SNIA focuses on the development of storage solution specification and standards. The association has shown an interest in the standardisation of storage platform technology used in Cloud Computing.

- **National Institute of Standards and Technology (NIST)** [180]: NIST is a non-regulatory US federal agency whose goal is the promotion of innovation and competitiveness through the advancing of standards and technology. Their focus has been on helping federal government agencies understand the benefits of Clouds and have created a formal definition of Cloud Computing as discussed in Section 2.3.

These standards should enable portability of data and applications across Clouds. The major focus of these organisations is on creating frameworks that enable two or more Cloud platforms to exchange information in a unified manor. However there is a potential issue that could arise from the number of organisations working on Cloud standards. The risk is that each standards organisation creates there own standard in isolation. This would leave a Cloud engineer in the same position as if there were no standards, where each provider could implement a different standard thus not improving interoperability.

There are a number of requirements that must be met to provide scalability when engineering on the Cloud. Increased resource utilisation should result in a proportional increase in performance. A scalable service should be capable of handling heterogeneity and scale across dissimilar Cloud hardware and software technologies. A service should be operationally efficient making use of only the resources it requires to service requests. A service should be resilient to hardware failure. Finally, as a service scales it should become more cost effective with resource pool growth.

The secure development of Cloud applications require a broad set of policies, technologies and control mechanisms to protect data, applications and infrastructure. Engineering secure Cloud platforms requires additional effort due to the use of Virtualization with specific concerns alluding to the potential to compromise the Hypervisor and gain access to other VMs running on the same host machine [273]. This is largely a theoretical concern at present but as the rewards of breaching this barrier increase with the adoption of Cloud Computing, so will the likely hood that exploits will be implemented. Engineering security requirements also cover identity management, privacy and legal issues that must be implemented with regulatory compliance already found in traditional enterprise IT environments.

Due to the complexity of the Cloud ecosystem and the possible reliance on multiple organisations when provisioning a Cloud service, an engineering requirement for man-

ageability is a must. This requirement refers to a number of workflow tasks relating to performance tuning, security compliance, disaster recovery and contingency planning. Manageability must be released at all levels of the Cloud Service Stack to ensure that resources and software are working optimally and interacting with End Users and other services correctly.

3.3.3 Web Scale Computing

The engineering of Cloud systems draws on the contribution of Web Scale Computing. Web Scale Computing, a relatively new term to software engineering, refers to the development of web based applications that scale with an ever increasing user base. The advent of "*web hubs*" and social networks, such as Facebook, Google and eBay, have seen the active development of new technologies to deal with the scalability limitations present in traditional distributed system technology. An example of which is the NoSQL [239] movement, where the issues of scalability that have plagued Database Management Systems (DMS) in web based environments have been overcome by re-thinking the technical requirements of storing data by obtaining a new compromise with data consistency.

3.3.4 Big Data

Issues with very large data sets, on an order of magnitude of a petabyte or greater, have recently contributed to the development of Cloud Engineering principles. These data sets are often refereed to as Big Data [25] recently defined by Mike2.0 [169] an open approach to Information Management. Big Data refers to large, complex data sets that cannot be captured, managed or processed by traditional on-premise computing systems and has driven the development of Cloud Computing technology to enable economically viable on-demand processing using third party computing resources. Cloud technologies, such as Hadoop discussed in Section 2.4.2.3, have reduced the complexities of engineering applications that analyse large data sets and improved the turn-a-round time of processing, enabling enhanced decisions that can be made sooner.

3.3.5 Design Patterns

Design patterns, as with other computer systems, play a role in enabling the reuse of generalised solutions to commonly occurring problems in Cloud Computing. There are a number of high level design patterns applied to Cloud Computing. These relate to fault tolerance, caching and monitoring:

- Design for failure: Avoid single points of failure and assume everything will fail.
- Caching: Cache data at network edges.
- Monitor Elasticity: What is not monitored cannot be controlled.

Designing for failure refers to the application of generalised fault tolerance techniques for the purpose of maintaining service availability. This is an active area of research [54, 67, 138, 244, 252, 279] and covers both compute and data resources in Clouds. Data replication and redundancy is discussed in detail by Zhang and Chen [279], where a number of strategies and their potential impacts are evaluated. For compute based fault tolerance, others [244] have discussed the use of High Availability (HA) as a system design approach. HA makes use of resource failover whereby standby resources are switched to in the event of a hardware component failing. In addition, system check points that save the state of a machine has been another avenue of research [67].

Due to the expense associated with the transfer of data in and out of Cloud providers, a design pattern has emerged to solve issues with geographically pervasive Cloud applications that leverage the use of caches [203]. This design pattern mandates the use of data caches at the edges of availability zones or data centres used by a Cloud application. This reduces the latency in accessing data after caching and enables the self-partitioning of data based of the frequency of use by an application. Additionally, caching overcomes the performance bottlenecks of a Cloud provider's interconnects that exhibit very high contention rations and are limited in bandwidth availability [176, 225].

Finally, the continuous monitoring of all aspects of the Cloud Service Stack has become a wide spread problem that is being solved by generalised design pattern solutions. These involve the tracking of resource usage patterns to enable elasticity and the collection of application level metrics [141] and is discussed later in Section 6.4.1.2.

3.4 Issues to Consider

Several elements need to be considered before any decision can be made on whether to migrate an application to the Cloud or make use of a Cloud provider for the development of a new service. This subsection highlights the advantages and disadvantages of using Clouds. In addition, methods and techniques that surmount to the current best practice in using Cloud technology are discussed. Finally, simulation tools for the purpose of ascertaining whether the performance provided by Cloud Computing is acceptable for a given application, are discussed.

3.4.1 Pros and Cons

There are multiple advantages and disadvantages migrating to the use of the Cloud Computing paradigm. This section provides an overview of the major pros and cons of using Clouds.

3.4.1.1 Disadvantages

Infrastructure and platform security is a major disadvantage of using Cloud Computing over in-house IT resources. The dependency on third party security measures limit the number of applicable use cases of Cloud Computing. Vendor lock-in is another disadvantage of using Clouds that can be overcome by selecting providers that make use of open standards. Hidden costs can be associated with using Cloud Infrastructure and need to be consider that are not present in traditional systems, such as the use of network bandwidth and metering associated with data storage. In addition, network latency needs to be evaluated for applications sensitive to response times, due to compute and network resource congestion and contention. The current track record of provider reliability can prove to be disadvantages as the length of service outages are outside the control of the End-User and can be detrimental to the profit of an application. The shared nature of Cloud resources can have an effect on privacy and measures must be taken to guarantee data is not exposed to the wrong people by safeguarding with the use of suitable encryption policies. Additionally, data protection laws on exporting data across jurisdictional borders, can be disadvantageous and problematic. Provider can go bankrupt, which must be mitigate by implementing suitable disaster recovery policies. There are cultural disadvantages that should be considered via the application of Change Management to mitigate the fear of job loss that outsource of IT services generates. Finally the dependency on network connectivity to deliver services can prove to be disadvantage if not foreseen during the development of an application and can limit usefulness. A good example of disadvantage over traditional applications is the Google Maps SaaS solution, which is not usable in remote locations with poor 3G coverage.

3.4.1.2 Advantages

On the other hand, the application of Cloud Computing can enable better collaborations between organisations as data is accessible from anywhere over the Internet. The economy of scale provided by a Cloud provider pertains to cost reductions and the inherent scalability of Cloud PaaS offerings enables elasticity to be achieved with minimal developer effort. The illusion of almost unlimited storage and computation resources reduce the

burden of dealing with the extremities of capacity management at the application development level. There are advantages to using Cloud providers for the backup and recovery of on-sight applications enabling the transfer of responsibility onto external *backup* experts, rather than training staff internally that can be costly and time consuming. Finally Cloud Computing brings a number of advantages with regards to the customisation of Cloud Computing services, access to Cloud services is available at multiple levels and can be exchanged with reduced development effort due to layers of abstraction applied.

3.4.2 Best Practice

Best practice refers to methods or techniques that consistently show superior results and describe standards that can be leveraged by organisations to successfully migrate to or adopt Clouds. As the Cloud Computing paradigm is still evolving best practice is still to be fully realised. The following list provides a number of high level best practices that can be applied throughout the life-cycle of Cloud software development and is inspired by research on the challenges Cloud Computing faces in Enterprise adoption [147, 148]:

- Adopt an appropriate iterative and agile methodology to enable the rapid deployment models of Cloud Computing to be leveraged by an application.
- Evaluate Cloud infrastructure suitability given the desired technology requirements of the application and the benefits provided by Cloud Computing.
- Create software unit tests for performance, security, reliability and availability to combat the shortcomings of Cloud Computing infrastructure and platforms.
- Adopt an integrated application life-cycle management tool to enable the benefits of Cloud Computing to be leveraged at all stages of the software engineering process.
- Adopt a daily build and continuous integration process to ensure software quality and provide early warnings of interface changes made by third party Cloud providers that could effect application stability.
- Automate the Quality Assurance (QA) of Cloud infrastructure to enable on-demand provisioning of resources to multiple providers with minimal human interaction and management.
- Automate regression testing to reduce software release cycle time so that changes in Cloud Computing technology have minimal impact on the overall stability of an application.

3.4.3 Simulators

Simulation of computing resources is a valuable tool when considering the performance and economical ramifications of using a Cloud provider. Modelling real Cloud systems and then running simulations can provide quantifications on the performance benefits of using different software architectures and resource allocation policies, in addition to insights on power consumption. Research on the simulation of distributed systems in not a new topic of research [120] but poses a number of challenges in the context of Cloud Computing due to the deployment of long lived services that do not fit with the job based approaches taken by Grid and other discrete event simulation software used to model parallel and distributed systems.

CloudSim [38], provides a framework for the modelling and simulation of Cloud Computing infrastructures and services. Its primary objective is to provide a generalised and extensible simulation framework that enables the seamless modelling, simulation and experimentation of emerging Cloud Computing infrastructures. Cloud developers using the framework are able to investigate design issues without concerning themselves with low level details. Some of CloudSim's limitations are discussed later in Section 7.3.3.

ICanCloud [126] is a simulation platform aimed at modelling and simulating Cloud Computing systems. Its main objective is to enable the prediction of the trade off between cost and performance given a set of application features. The platform provides a number of configurations out-of-the-box for modelling Cloud storage systems including local storage, remote storage and distributed parallel storage systems that leverage Distributed File Systems (DFS).

Other simulation tools that can be used to model Clouds are JavaSim [134] and its related native implementation C++SIM [134] and SimJava [120,228]. Although not specifically tailored to support the modelling of Cloud environments they do provide generalised frameworks for distributed system modelling using discrete event process oriented simulation.

3.5 Configurable Parameters

The content within this Chapter has highlighted a number of parameters that are configurable and can be dynamically updated within a Cloud. The parameters can cause problems if they are not considered by Cloud software or application developer. The more interesting of which have been summarised as follows:

- **Discoverability**: The operation time discoverability of services and resources for

use by a Cloud application can have a dramatic impact on QoS. Due to the dynamic nature of Cloud resources, if an application moves from one Cloud environment to another it must be able to discover services and resources that will maintain any agreed QoS level.

- **Data Locality**: The physical storage location of data of which a Cloud application makes use of can be problematic due to legal restrictions on data jurisdiction. If this parameter is not considered or configured correctly within a cloud application as it moves around its environment there maybe the possibility of a breach in SLA with legal ramifications.

- **API Selection**: Issues of dynamic binding and API selection with regards to Service Orchestration, if not considered by an application developer, have the potential to limit the interoperability of a Cloud. This in turn could limit the obtainable QoS an application can reach if it is migrated to a provider and not correctly configured to make use of the provider's localised high performance Cloud services, such as distributed block storage.

These parameters are of interest as they have an impact on the adoption and ease of use of Cloud Computing. Some of these parameters are discussed in further detail as part of Chapter 6 and the topic of Contextualization.

3.6 Summary

To summarise, this chapter of the thesis has discussed the relevance of SOA architectures in Cloud Computing and the roles service orientation and orchestration play in the composition of Cloud applications developed as services. The topic of Web Services and the technology used to create them have been presented. The topic of Cloud Engineering, the challenges it faces and requirements needed for a systematic approach to the development of Cloud applications, including the standards organisation that are facilitating in this process, were discussed. The heritage of Web Scale Computing and Big Data in Cloud Engineering was introduced and a number of design patterns enabling the reuse of solutions to regularly experienced problems in the development of Cloud applications were presented. Finally, a discussion was held on the roadblocks in place that need to be overcome before applications can be migrated to and adopted by Cloud Computing. This included the relevance of simulation tools for evaluating application performance and a number of simulator implementations were presented. Furthermore the pros and cons of

using Cloud technology from the point of view of an Enterprise where discussed, including best practice for the effective development and composition of Cloud applications.

Chapter 4

Grids on Clouds

4.1 Introduction

The previous chapter discussed the development and composition of applications in Cloud Computing. This chapter introduces the paradigm of Grid Computing and its uses in supporting the development of scientific applications. Parallels are drawn between the technologies developed in Grid Computing and their applicability to Cloud Computing. In addition, many of the applications that run on Grids have similar requirements to those executed on Clouds. It is apparent that there are many similarities between the two paradigms and much can be learnt from the past mistakes made in Grids. This background on Grid Computing presented herein, sets the scene for the research of Chapter 7 on using Cloud resources for Grid systems.

This chapter places emphasis on the non-trivial QoS enabled by the resource management of Grid Computing and how this can facilitate the development of the Cloud paradigm. Extensive research has been performed on the use of Service Level Agreements in Grids but its usefulness has not been fully appreciated as the Grid Economy was never realised. This however could change, as the importance of providing better than best effort QoS in Clouds creates opportunities for competitive advantage and the ability for a provider to distinguish itself from the competition.

The remainder of this chapter is structured as follows: Section 4.2 introduces the topic of Grid Computing providing a definition and discusses the architecture of Grids inclusive

of the applications that make use of them. Section 4.3 discusses the concept of resource management in Grids including the relevance QoS plays. Additionally, resource allocation and the monitoring of Grids are discussed. In Section 4.4 the importance of Service Level Agreements and the technology enabling their application in Grid environments are presented. Section 4.5 introduces Grid Middleware and a number of popular implementations. Finally, in Section 4.6 some of the differences between Grids and Clouds are highlighted and the idea that theses two paradigms can be complementary is put forward.

4.2 Grid Computing

The term *Grid Computing* originates from the 1990s as a metaphor for making computing power as accessible as electricity from power grids. The evolution of Grid Computing from Cluster Computing came about with a need for organisations to share their computing resources and was enabled by the introduction of middleware designed to support data intensive applications over wide-area infrastructure [77]. An early definitions of Grid Computing defined it as:

> "... a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities." [146]

Grid Computing is a paradigm in distributed systems that enables computing resources from multiple administrative domains to be combined into a federation for the purpose of reaching a shared common goal. A defining characteristics of a Grid over a traditional cluster, is the trend towards loosely coupled heterogeneous and geographically dispersed resources. A more concrete definition of Grid Computing was presented in the seminal work of Foster et al. with a simple three point check-list [81]. Accordingly a Grid System is:

1. A system that coordinates resources not subject to centralised control.
2. Uses standard, open and general purpose protocols and interfaces.
3. Delivers non-trivial qualities of service.

A Grid system thus integrates and coordinates resources that are within different control domains. In addition, a Grid is built from multi-purpose protocols and interfaces that address fundamental issues of authentication, authorisation, resource discovery and

resource access. Finally, a Grid must enable the constituent resources to be coordinated in a away to deliver various degrees of QoS relating to response time, throughput, availability and security. Therefore the utility of the combined resources of a Grid must be significantly higher than the sum of its parts.

4.2.1 Architectures

Early architectures of Grids organised components into layers [82]. Each layer shared a common set of characteristics. These layers are:

- **Fabric Layer**: Provides access to physical or logical shared resources from computational, storage or network hardware. This layer enables the discovery of resource structure, state and capability.
- **Connectivity Layer**: Provides core communication and authentication protocols enabling the exchange of data between resources. Authentication provides secure mechanisms for validating access to resources and include functionality to enable single sign-on across all organisations partaking in the federation, delegation of credentials for running applications on behalf of a user with out granting that user direct access to the resource and integration with local security solutions.
- **Resource Layer**: Defines protocols for the secure initiation, control, accounting and monitoring of shared resources that is split into two primary protocol stacks for information sharing and resources management purposes.
- **Collective Layer**: Provides protocols to manage the interactions of resource collections across administrative domains, enabling a wide number of sharing policies without defining additional requirements on shared resource pools.
- **Application Layer**: A container layer for the user's application that provides access via APIs to services defined in any of the other layers.

Later standardisation by OGF defined an open standard for Grid Architectures based on Web Service technology. The standard was named the Open Grid Services Architecture (OGSA) [123] and leveraged SOAP and WSDL. The standard defines a set of extensions on the use of WSDL to enable stateful Web Services, defining approaches to creating, naming and managing the lifetime of a service instance. In addition, it provides mechanisms for asynchronous notifications and handling of service invocation faults.

4.2.2 Applications

There are a number of application types that are run on Grids. Many of the application types are highly related to those that are executable on current Cloud providers, as outlined in Chapter 2. In Grids, four distinct classes have been defined and described by Foster et al. [146] and are as follows:

- **Distributed Supercomputing**: Applications that solve large scale computationally expensive problems via the use of many thousands of machines to obtain higher processing power.
- **Real Time**: The processing of real time data sources that rely on distributed storage and network caches for improved performance and reduced latency.
- **Data Intensive Computing**: Applications that focus on the processing and analytics of large datasets.
- **Tele-immersion**: Enabling users in geographically distributed sites to collaborate in real time via a shared simulation environment.

Additionally, a taxonomy of Grid Applications is presented in [241] that classifies an application on the concepts of control and data parallelism used. It is also worth noting that applications often fit into multiple categories but by providing a clear taxonomy, an estimation of potential performance gains can be found for applications that are not already executable in a Grid environment.

4.3 Resource Management

The resources that are present in a Grid are often distributed geographically around the globe and reside in different administrative domains. The following sections discuss the impact QoS has on the allocation of resources in Grids and relates this to the management of resources in Cloud Computing.

4.3.1 Quality of Service

By exploring the current state of QoS in Grid Computing, the lessons already learnt can be exploited and potentially utilised in Clouds. In the early 21st Century the dynamics of the Internet economy changed and the ratification of e-commerce as a new source of revenue growth within businesses appeared. This led to the development of the Web Service, as

businesses turned to SOAs to simplify their interactions in the digital world, through the loose coupling of service providers and consumers.

The introduction of Web Services affected the development of Grid Computing as emphasis was placed on Grids providing services to reduce the complexity and cost that had been previously associated with them. The Service-Oriented economy also provided the mechanism to create virtual organisations where computation resources could be shared securely. Service oriented Grids created new problems concerning the management and availability of shared resources across organisational boundaries. Grids relied for many years on the provisioning of resources on a best effort guiding principle of operation and as interest in commercial utilisation of Grids surmounted, more stringent guarantees on the management of resources via QoS were realised as a necessity for the wide spread adoption of Grids to take place in industry [217].

The following subsections focus on the management of resources to guarantee performance and the technology in place to facilitate the reservation of resources, both of which are highly relevant to Cloud Computing.

4.3.2 Resource Allocation

Without the management of resources Grids would be unable to function. Resource management encompasses the dynamic allocation of tasks to computational resource and requires the use of a scheduler (or broker) to guarantee performance. QoS is enabled in Grids by the efficient scheduling of tasks, this guarantees that resource requirements of an application are strictly supported but resources are not over provisioned and used in the most efficient manor possible. Sequences of tasks are represented as workflows, directed graphs comprised of precedent constrained nodes, which each represent the specific ordered invocation of a service on computation resources to process a given task. Several research projects have tackled the complexities of resource reservation and allocation in Grids such as the Phosphorus Project [197] and utilise schedulers such as DSRT [44] and PBS [196]. Research within Grid Scheduling is still an area of activity [161].

An example of a Grid software stack enabling resource management is The Globus Toolkit [88]. It has become the academic and industry leading open source software solution for building Grids and provides the necessary middleware to manage and monitor resources. Simulation and modelling has furthered the understanding of Grid Architectures and will do so in Cloud Computing. Advancements in simulation and modelling techniques will aid in the better understanding, usability of and streamlining of Cloud environments, as was seen in the development of Grids. Progress is already being made

towards the development of a simulation tool. Currently the Grid Computing and Distributed Systems Laboratory from The University of Melbourne has identified the need for such a tool to support the performance evaluation of Cloud environments and is in the early stages of development, named CloudSim [38]. The simulator is based around the programming framework they previously created to model Grids, in the Grid simulator, GridSim [106].

4.3.3 Monitoring

Monitoring tools are essential in ascertaining the availability of resources and providing feedback to schedulers within Grids. Monitoring tools enable guarantees to be made on the performance of any given resource by making sure that the computational resource in question is not over utilised and is on-line. Performance is characterised by the amount of useful work accomplished by a computer system in comparison to the time and resources used. Monitoring tools are also essential in providing fault tolerance and the migration of tasks in the event of a resource failure in the Grid. Fault tolerance involves the identification of a resource failure via monitoring tools, the rescheduling of the task, that was running on the failed resource, to an alternative available resource and migration of the state snapshot of the task to the newly allotted resource, at which point the task continues execution [221, 227]. The state of a task in execution must be regularly saved for fault tolerance to function, this process is known as check pointing. Many monitoring tools have been developed for Grids [129, 163, 232, 243, 246].

4.4 Service Level Agreements

As the importance of Service Level Agreements (SLAs) as facilitators for the widening commercial uptake of Grids has grown, substantial effort has been made in standardising their use. The Web Services Agreement Specification (WS-Agreement) [9] is one such standardisation effort by the Open Grid Forum [183]. WS-Agreement is a Web Services protocol for establishing an agreement between two parties, using an XML for specifying the content of an agreement and agreement templates used to discover appropriate parties. The specification consists of three parts:

- A schema for specifying an agreement.
- A schema for specifying an agreement template.
- A set of operations for managing an agreement's life-cycle.

Although WS-Agreement can be effectively used to facilitate SLAs, the life-cycle model does not accommodate the dynamic nature of the Grid economy, providing facilities to negotiate and renegotiate an agreement. The current state of the art research in QoS within Grids is concentrating on this problem [59, 223]. Another cutting edge area of research surrounding QoS in Grid Computing, is solving problems related to risk assessment and dependability of service providers and is being tackled by projects such as AssessGrid [60]. The AssessGrid Consortium [17] have researched heavily into QoS but more specifically SLA's. Many of the objectives of the project are also relevant in the context of Cloud Computing, such as how to evaluate the reliability of Cloud service providers and how best to estimate the risks involved in accepting any given SLA.

As with past and present Grid Computing projects [105, 230, 231], SLAs will play a major role in the development of the Cloud Computing paradigm. Within the research topic of QoS in Clouds, emphasis will have to be placed on the performance and availability of Virtualisation technology and the tools necessary to monitor virtualised hardware. Facilitating QoS in Cloud Computing is an area of intense research and is part of the research aims and objectives of the open source, research backed Cloud implementations discussed in Section 2.4.

4.5 Grid Middleware

Grid Middleware has been defined as a software layer within a distributed system that enables transparent access to heterogeneous computing environments that reply on different platform technology and networking protocols while transcending administrative boundaries [162]. The following subsections introduce a number of popular Grid Middleware in active use:

4.5.1 Globus Tookit

The Globus Toolkit [78, 88] is an open source toolkit, implemented in `Java` and `C`, for building Grids. It is being developed by the Globus Alliance [89]. The toolkit includes software services and libraries for resource monitoring, discovery and management in addition to security and file management. The tools are packaged as a set of components that can be used independently or together to develop a Grid based application. Further details on the architecture of the Globus Toolkit are provided in Section 7.2.

4.5.2 UNICORE

UNICORE [218, 250] or "*Uniform Interface to Computing Resources*" is a Grid Middleware that offers ready to run Grid systems that make distributed computing and data resources available seamlessly and securely over the internet. It is an open source project implemented in `Java` that is based on a number of open standards by OGF including OGSA. In addition it provides workflow support that is tightly integrated into the UNICORE software stack, while still being extensible in order to use different workflow languages and engines for domain-specific usage.

4.5.3 gLite

The gLite [154] middleware provides a framework for the development of applications that make use of resources distributed over the internet and is comprised of an integrated set of components designed to enable resource sharing. It was produced by the Enabling Grids for E-SciencE (EGEE) [66] project and pulls together contributions from many other projects.

4.5.4 GridWay

GridWay [107] is an open source meta-scheduling technology that enables the large scale, secure, reliable and efficient sharing of resources managed by different distributed resource management systems such as PBS. It provides a user with a scheduling framework similar to local distributed resource management command line interfaces they have become familiar with while enabling job execution across heterogeneous, dynamic and loosely coupled Grids.

4.5.5 Oracle Grid Engine

Oracle Grid Engine [87] is an open source batch queuing system developed by Sun Microsystems (recently acquired by Oracle) that is typically used on HPC clusters for the purpose of accepting and scheduling standalone and parallel jobs.

4.5.6 Other Grid Middlewares

The above list is by no means exhaustive. There are number of other Grid Middlewares that are discussed as part of a taxonomy on Grid resource management systems for distributed computing in [150, 162].

4.6 Grids Vs Clouds

The advantages of using Cloud Computing in Grids have motivated interest in running Grid Middleware on IaaS to leverage the many years of research that have already been performed on the subject area of Grid Computing. Research on this topic first started out by defining the similarities and differences between the two paradigms. Forster et al. were one of the first to describe the core differences between Grids and Clouds [80]. In this work they describe how aspects related to security, the programming model, the business model, applications run and levels of abstraction used are different. In addition, Buyya et al. discussed the parallels of research on the yet to be realised Grid Economy and market oriented allocation of resources within Clouds [219]. Table 4.1 provides a high-level view of some of the major similarities and differences between Grids and Clouds, inspired by this research:

Table 4.1. High Level Comparison of Grids and Clouds.

Feature	**Grid**	**Cloud**
Resource sharing	Collaberative (VOs)	Assigned resources not shared
Virtualization	Data and Comptuing resources	Hardware and software platforms
Secuirty	Credential delegations	Isolation
Architecture	SOA	User chosen
Software Dependencies	Application domain dependent	Application domain independent
Platform awareness	Client software must be Grid enabled	Customised environment
Scalablity	Nodes and sites	Nodes, sites and hardware
Management	Decentralised	Centralised
Usability	Hard to manage	User friendly
QoS guarantees	Limited Support	Limited Support

However this research does not consider that the two paradigms can be complementary and that the technology can be combined to bring the benefits of both, the federation of resources across administrative domains and the economical advantages of on-demand provisioning. There are however a number of issues when contemplating the running of Grid Middleware on Cloud IaaS providers and this brings a number of interesting research

challenges that must be overcome along side a number of technical limitations. These are discussed in detail within Chapter 7.

4.7 Configurable Parameters

The content within this Chapter has highlighted a number of parameters that are configurable and can be dynamically updated within a Cloud. The parameters can cause problems if they are not considered by Cloud software or application developer. The more interesting of which have been summarised as follows:

- **Collections**: The dynamic configuration of Cloud resources to form collections is a necessity born out of the need for a Cloud application to scale on-demand. If careful consideration is not given to how an application can be made self-aware through configurable resources related parameters, it will be unable to balance its workload across available resources and maintain its QoS.
- **Connectivity**: The network connectivity between cloud resources enables the formation of clustered resources. If care and attention is not placed on the configurable parameters of an application's network stack as its environment changes dynamically, resources and service dependencies will be unavailable and have an overall impact on the application's availability.
- **Monitoring**: The monitoring of Cloud resources is critical to reacting to unforeseen events, such as flash flood traffic, that could have an impact on an applications QoS. As a Cloud application moves between Cloud environments, the reporting of monitoring metrics may need to change for performance reasons. If this locational parameter is not considered, QoS cannot be effectively monitored and controlled.

These parameters are of interest as they have an impact on the adoption and ease of use of Cloud Computing. Some of these parameters are discussed in further detail as part of Chapter 6 and the topic of Contextualization.

4.8 Summary

This chapter has discussed the topic of Grid Computing providing insights into the architecture of a Grid and the applications that they execute. The concepts behind resource management in Grids was presented including its relevance to maintaining QoS when

provisioning shared Grid resources. Additionally, resource allocation and the monitoring of Grid infrastructure were discussed. The importance of Service Level Agreements and the technology enabling Grid environments to provide better than best effort QoS was presented. A definition of Grid Middleware was discussed and a number of Grid Middleware implementation in active use within the scientific community were presented. Finally, some of the differences between Grids and Clouds were highlighted and the idea that theses two paradigms can be complementary was put forward for later discussion.

Substantial effort has been placed on the performance of Grid Computing [4,133,243]. The same can be expected for Cloud Computing, in the expectation that users are going to adopt this paradigm. In the next chapter the performance of Cloud infrastructure is evaluated and a number of performance issues are presented.

Chapter 5

Cloud Infrastructure Performance

5.1 Introduction

The main contribution of this Chapter lies in the findings of a performance evaluation into a specific aspect of Cloud Computing, the management and use data at the Virtualization level. Results are presented that provide incite into how Cloud technology can be improved and what technology is best for the needs of a given application.

After decades in which companies used to host their entire IT infrastructures in-house, a major shift is occurring where these infrastructures are outsourced to external Cloud providers. Maintaining the elastic nature of resources when provisioning computational, storage and networking services has thus far been a major concern of the providers. For the Cloud Computing paradigm to survive in an increasingly complex information world, the requirement to provide increasingly efficient services is becoming more prevalent. Underpinning a successful Cloud infrastructure is the delivering of a specified QoS to its users.

QoS is a broad topic in Distributed Systems and is most often referred to as the resource reservation control mechanisms in place to guarantee a certain level of performance and availability of a service. The research scope is usually concerned with the management and performance of resources such as processors, memory, storage and networks in Cloud Computing. A SLA specification usually provides a formal method for describing QoS requirements. Services (e.g. compute, storage, database) are generally

offered as pay-as-you-go plans and hence have become attractive to many customers.

Infrastructure as a Service (IaaS) providers play a pivotal role in the QoS provisioned in the majority of Cloud architectures comprised of an interchangeable multilayer software stack [266]. Virtualization, as the fundamental resource building block of IaaS, is critical to maintaining acceptable levels of performance to prevent breaches in SLAs, thus increasing the overall profitability of a Cloud [280]. This provides primary motivation for efficient Hypervisor design and remains a limiting factor in what applications are deployable and can take full advantage of a Cloud. Although research on the topic of Virtualization is not new, it has seen a resurgence of interest in recent years in the problem domain of Cloud Computing [127, 138, 281]. Two such areas of interest are: VM lifecycle management [92, 115], where guarantees that a VM will be on-line within a certain time-frame are of importance to the rate at which IaaS can react to changes in demand [253]; and I/O scheduling [145], which can have adverse affects on application performance [63].

This chapter compares, via benchmarking, two Cloud infrastructure managers Nimbus [140] and OpenNebula [236] alongside two open source Hypervisors XEN [21] and KVM [153] using a combination of synthetic benchmarks. The aim of this work is to evaluate and expose the limitations of these technologies via results that provide insight into the rapidly evolving landscape of Cloud tools. The core contribution of this chapter lie in the findings of a quantitative evaluation into the performance overheads of:

i) VM image formats

ii) Propagating VM images to physical resources, at the IaaS layer

iii) Paravirtualized I/O devices used to access the data held within an image after propagation

A taxonomy of Virtual Machine image standards, inclusive of features and Hypervisor support, is presented. The chapter demonstrates that overheads have the potential to influence the performance of a Cloud and thus the usage patterns of IaaS providers by its users.

In addition, an argument is presented that previous work on the subject of Hypervisor I/O performance is now outdated due to the pace of development surrounding Paravirtualized device drivers. A conclusion is drawn that reducing overheads could lead to an increase in the pace at which Cloud Computing is adopted in industry. Firstly a reduction in resource acquisition waiting times would enable quicker reaction, by an IaaS, to a changing environment. Secondly an increase in the performance of Paravirtualized I/O

devices would reduce operating costs via a correlated reduction in the number of virtual resources needed to be provisioned to service a given number of client requests.

5.1.1 Performance Issues in Clouds

It is difficult and most likely impossible to express QoS as a single scalar value indicating the ”goodness” of a service, because the context is of large importance to the interpretation of any assured QoS. The literature describes various application-level contexts where an estimation (and thus quantification) of quality is desired and required, for example: Performance, Dependability, Reliability, Scalability, Capacity, Robustness, Manageability, Availability, Interoperability, Security as well as many more **bilities*. Some exhibit precise quantifiable definitions, others do not (e.g. Interoperability) or are super ordinate concepts of other attributes. Furthermore, there are dependencies between attributes (e.g. Scalability and Performance), thus a clear separation is not always possible.

Clouds have pushed *Virtualization* into the lime light as a new technological requirement. This has only recently become feasible because of the performance enhancements that have been made to Virtualization hardware and software technology. This has improved the performance frontier of virtual machines to nearly that of the underlying hardware they run on, achieving near native performance of the virtualised resources exposed within a virtual machine, through a reduction in the overheads associated with context switching physical resources between the virtual machine instances.

Without the *management of resources* Clouds would be unable to function. Resource management encompasses the dynamic allocation of tasks to computational resources and requires the use of a scheduler (or broker) to guarantee performance. QoS is enabled in Clouds by the efficient scheduling of tasks, this guarantees that resource requirements of an application are strictly supported but resources are not over provisioned and used in the most efficient manor possible.

Distributed storage like in Grid Computing and other distrusted paradigms plays a large role in the scalability of Cloud Computing. Some software services based around the map/reduce functional programming principles currently provide access to large amounts of replicated scalable storage through a simple high level interface. Further research is still required in Cloud storage services as the efficiency and performance of data storage and management can become a bottleneck in a distributed system and thus effect QoS.

Scalability through a federation of Clouds and multi-clouds is another issue that brings its own performance questions. In a typical scenario, several Infrastructure Providers (IP) establish collaboration in which any IP can rent capacity from the others and also allows

these to use its own capacity, according to internal IP business policies. This can impose constraints onto the IP regarding performance, e.g., with regards to the affinity of service components.

This list of issues is by no means exhaustive. Other issues such as *green assessment* is also worth considering in the big picture. The development of bottleneck analysis tools are needed to assess potential ecological performance measures and how these measures are affected by changes in workload, configuration, and infrastructure utilisation.

This research focuses on IaaS platforms. IaaS roots can be traced to Grid Computing. In Foster's seminal work [82], the infrastructure was called Fabric and comprised both computational and storage resources. Grid registries such as the Index service of the Globus Toolkit [79] allowed the discovery of specific types of computational and storage resources with queries based on properties representing their static and dynamic characteristics. The evolution of Grid research brought the concept of Virtual Workspace (VW) [140]. The Globus implementation of the VW concept, the Virtual Workspace Service, was at the core of what is now called Nimbus [177] and is described as an open source toolkit that enables a cluster to be turned into an IaaS Cloud. Nimbus is a Virtual Infrastructure Manager (VIM) with a Web Service provisioning interface. Another open source VIMs considered in this Chapter is OpenNebula [236], an implementation of the research being performed and led by Reservoir [209], a European Union FP7 funded research initiative in virtualised infrastructure and Cloud Computing.

5.1.2 Related Work

There have been numerous studies on the performance of Hypervisors within the literature [58, 135, 145, 193, 238, 278, 285], this research differentiates from these by giving a contemporary performance evaluation within the context of Cloud Computing. In [206], performance results of some of the image types supported by KVM are presented but not all and not compared to another Hypervisor such as XEN, as evaluated as part of this research. A benchmark is contributed and implemented to evaluate end to end image propagation overheads of any IaaS. [283] present a performance analysis framework for the overheads in resource acquisition and release. However the results of their work are for resources deployed onto Amazon EC2 with the framework left untested on other virtual resource management architectures. [92] describe a proof-of-concept framework for facilitating resource management in service providers, in which a performance evaluation of the overhead of creating a VM image is considered, not the time to propagate an already existing VM image to a physical resource. [42] explore how multicasting file

transfers can reduce image propagation overheads.

Previous related work have concentrated on the performance and scalability of CPU Virtualization [193], omitting a performance analysis inclusive of I/O. An additional publication [238] formulates an approach to performance evaluation, using real world workloads on Amazon EC2 and considers the system as a "black box" without finding the root causes of the performance bottlenecks. Another creates a methodology for the collection of virtual I/O metrics [285] but evaluates the performance of only a single Hypervisor. Other publications [135, 278] incorrectly setup a parameter of the IOzone benchmark, using a 64MB test file instead of twice the size of available memory, bringing into question the validity of the results obtained, an issue rectified by this research. Figure 5.1 shows results recorded when using a 64MB test file and how incorrect parameter selection provides meaningless results as all that is measured is the speed of machine's RAM. [58] present results that KVM out performs XEN in IOzone tests. The experiments of this work have been unable to confirm these results and one can only assume, due to limited information from the paper on the experimental environment used, that again the file size parameter for the IOzone benchmark has been set incorrectly. Benchmark parameter selection is discussed in further detail as part of Section 5.4.2.

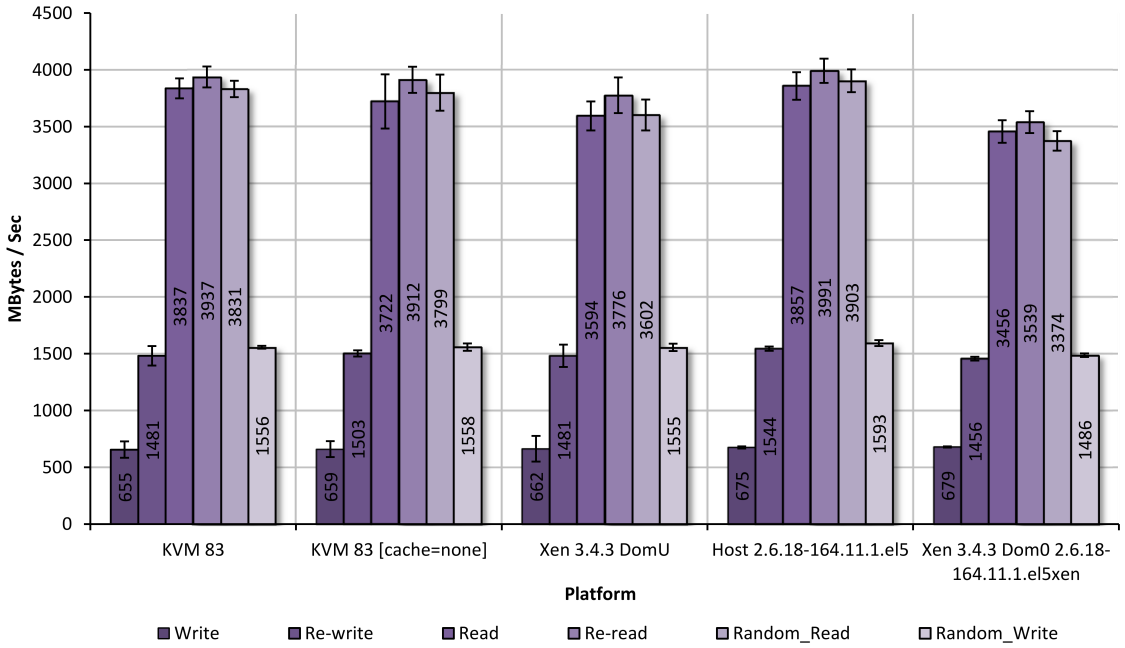


Figure 5.1. Effects of Incorrect Parameter Selection

This remainder of this chapter is organised as follows: Section 5.2 discusses and compares the propagation of images by Virtual Infrastructure Managers. Section 5.3 discusses the topic of virtual machine management, two open source Hypervisors Xen & KVM and

their support for Paravirtualization. Finally, Section 5.4 presents performance evaluations on virtual machine image formats, image propagation delay and Paravirtualized block I/O.

5.2 Image Propagation

The following subsections discuss the life-cycle and propagation of images in the context of two open source toolkits, which can be used to create an IaaS Cloud. The toolkits: OpenNebula [188] and Nimbus [177], manage VM images and physical resources to create virtual resources. An overview of each is provided with insights into the heritage and design of the toolkits. In addition, analysis on the mechanisms used to propagate VM images and the source of potential delays or overheads are discussed in relation to design decisions.

5.2.1 Virtual Infrastructure Management

An electrical utility provider requires systems to be in place to provision and monitor resources for the purpose of ensuring service reliability and performance to keep pace with changes in demand so that consumer usage patterns and high resource contention do not adversely affect the QoS provided. Cloud IaaS providers are similar in that they use Virtual Infrastructure Managers (VIM). Virtual resources need to be brought on and off-line as required and monitored to assess status so that intelligent decisions can be made on how best to use the underlying physical resources for the business objectives of an IaaS provider. VIMs achieve this through a scheduling component that assigns VMs to physical resources with feedback gathered from monitoring services from both the physical and virtual infrastructure. A scheduling component orchestrates with others in the system to:

i) Assess the needs of a VM

ii) Provision a suitable physical resource

iii) Transfer a VM image from a central repository over the network via an available network protocol

Once transferred a Hypervisor adapter component is used to execute the VM image on the physical host machine and bring the virtual resource on-line.

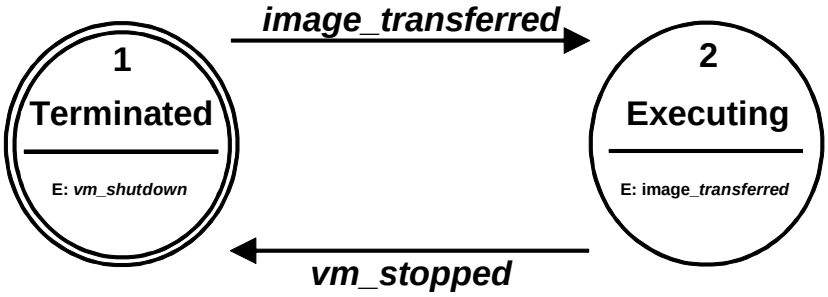


Figure 5.2. Simplified life cycle states of a non persistent VM

In the life cycle of a VM managed by an IaaS, overhead is associated with transitions of state illustrated in Figure 5.2. IaaS requires time to provision resources, transfer data and confirm termination. A trade off exists where at some point reducing these overheads incurs a penalty associated with polling and the contention of resources. This is explored in Section 5.4.4.2.

5.2.1.1 OpenNebula

OpenNebula [236] is an implementation of the research being performed and led by Reservoir [209], a European Union FP7 funded research initiative in virtualised infrastructure and Cloud Computing. It can be used as a Virtualization tool for the manipulation of local virtual infrastructure within datacenter clusters for the creation of private Clouds. Public Cloud support, via a selection of management interfaces, exposes functionality of a remote provider's VM, storage and network resources using de facto standards through a locally accessible portal. Hybrid Clouds, made possible through the combination of public resources and local virtual infrastructure, enable highly scalable application hosting environments. The design of OpenNebula efficiently manages multiple types of workloads with emphasis placed on addressing business requirements and use cases. A black box approach to design facilitates the management of VM Operating Systems (OS) and services agnostically thus allowing the system to be used in HPC environments.

The modular design and implementation of OpenNebula allows for ease of extension and support for plug-in replacement features and functionality. Cloud plug-ins enable access to public provider resources, such as Amazon EC2 and ElasticHosts using dissimilar interfaces, enabling support for federated Clouds and simultaneous access to several remote Clouds. Libvirt [27], a Hypervisor agnostic API, is used to provide access to multiple Virtualization technologies via a single interface. Policy based scheduling capabilities can be swapped for advanced reservation of capacity via Haizea [235], a VM lease

manager that can replace and override the standard scheduling component shipped with OpenNebula.

5.2.1.2 Nimbus

Nimbus [140] builds on past research into Grids by reusing many of the standards and technologies invented by the Grid community, particularly those used in the Globus Toolkit. It provides an upgrade path to the Cloud for organizations using Grids. This is enabled via the features of Nimbus that make use of Grid resources and its integration with familiar resource schedulers, such as PBS, to schedule VMs. Development of Nimbus has placed emphasis on use cases applicable to the needs of scientific community but many non-scientific applications are still well suited to the virtual infrastructure environment it provides.

The components of Nimbus are modular in design with Hypervisor agnostic support again enabled via Libvirt and support for Amazon EC2 via a SOAP based API for invocation and securing of off-site resources. Nimbus also supports WSRF frontend for controlling virtual infrastructure and context broker and agent components enabling automated contextualization of VM images for "one-click" clusters [142]. Nimbus provides interoperability with existing Grid Security Infrastructure (GSI) using Public Key Cryptography (PKC) to access resources.

5.2.1.3 A Comparison

As the use cases of the two VIMs evaluated in this research differ so do the availability and implementation of features. Experience with the two systems have highlighted some of these differences and can be found in Table 5.1. IaaS neutrality describes the capacity of a VIM to be able to connect to third party Cloud providers. OpenNebula supports a selection of plugins while Nimbus has limited support for providers other than Amazon. This could change as additional backend APIs become available to access other providers. An explanation for this could be that more development effort has been assigned into seamlessly integrating Nimbus with existing Grid infrastructures or that the more advanced contextualization methods used complicate integration with third party Clouds.

Table 5.1. Comparison of the VIMs

Feature	**OpenNebula**	**Nimbus**
IaaS Neutrality	Support for using other providers via Cloud plugins	Limited support for using other providers
Contextualization Method	Boot time initialization bootstraps context via generated ISO CD image	Context agent contextualizes over network using context broker service
Security Context	Support for embedding certificates at run time, eliminating potential security risks	
Image Pervasion	Minimal alterations needed to contextualize image	Pervasive contextualization necessary due to context agent
Image Propagation Process	Limited support for image persistence via a stage out mechanism	Support for both staging in and out images
Networking Support	Scripted network contextualization via IP pools and IP to MAC binding. Limited support for multi-site migration.	Support for cross site networking and multi-subnet environments.
VMM Support	Support for popular Hypervisors via Libvirt: XEN, KVM, VMware	

The VM contextualization facilities provided by both systems use similar ideas but differ in their implementations. Both OpenNebula and Nimbus support contextualization of VM images at boot time, setting the context of a VM instance such as IP address, hostname and security access policies. The contextualization process in Nimbus is potentially more powerful in the respect that a centralized broker service coordinates VM instances and can control the context of multiple virtual clusters, even during execution. The security context of the virtual cluster is set through the generation of security keys, using PKC, performed at runtime, permitting access to the virtual resource without storing private keys within a VM image.

The contextualization of virtual clusters in OpenNebula differs in that each VM image

requires an additional ISO CD image be mounted and an embedded script be executed at runtime. This enables nodes of a cluster to reuse the same VM image with small alterations to a VM description file, used to generate an ISO CD image with service, network and security context. The use of an ISO CD image could be seen as less pervasive than in Nimbus as no installation of a context agent into a VM image is required for the contextualization process.

The implementation of the image propagation processes used in both systems differ. OpenNebula provides a facility to persist VM images once staged in and after execution on a remote host but does not provide a facility to return the VM image to the image repository or a stage out process. Nimbus on the other hand provides both staging in and out of images from and to an image repository as well as a mechanism to integrate with the GSI to allow the secure propagation of images from an off-site location to an on-site repository.

The systems have varying support for different types of network topologies. OpenNebula does not provide automated virtual networking but instead provides networking configuration scripts for both dynamic and static network environments. This enables an IP to be paired with the MAC address of a VM assigned when the VM is contextualized and from an IP pool. This prevents a VM from being assigned to another IP pool and thus a VM cannot be migrated across network subnets, limiting the ability to have truly federated Clouds but simplifies the contextualization process of the VM. Nimbus however provides a mechanism to assign an IP to a VM via the context broker, as needed during runtime, using a localized DHCP server and the filtering of network packets. This enables a VM to be reassigned an IP on the fly across networks.

While there are differences between the two systems, there are also some similarities. Both systems have adopted Libvirt as a Virtualization API so that development can concentrate on the core of the projects managing virtual infrastructure rather than maintaining support for volatile and rapidly evolving Hypervisors. Both systems exhibit a module design that permits a certain degree of flexibility when creating extensions and provide support for using external resources for the creation of Hybrid Clouds.

The VIMs make use of two transfer protocols: SCP and Grid Security Infrastructure File Transfer Protocol (GSIFTP). SCP provides a means of securely transferring files between two machines using a Secure Shell (SSH) tunnel. The SSH network protocol provides several forms of encryption cipher and authentication methods. GSIFTP is a subset of the GridFTP protocol, provided in the Globus Toolkit, which is an enhancement to the standard FTP protocol that enables authentication through the GSI via PKC. It does not include many features of the GridFTP protocol. For instance parallel data transfers,

enhanced reliability or automatic TCP window and buffer sizing. Nimbus out of the box provides facilities to use both SCP and GSIFTP while OpenNebula only provides SCP as a file transfer protocol; an evaluating all three configurations is provided in Section 5.4.4.2.

5.3 High Performance Virtualization

A computer system can be virtualised in a number of ways that can impact on performance. This section of the thesis discusses the different techniques used to virtualise a guest operating system, provides a comparison of two open source Hypervisors in light of these techniques and introduces the concept of block I/O Paravirtualization for high performance data intensive Cloud applications.

5.3.1 Virtual Machine Management

To recap, a VMM or Hypervisor partitions a physical host machine though the use of three generalized techniques: Full Virtualization, Paravirtualization and HVM and is responsible for controlling the life cycle and resource utilization of local VMs. These techniques provide a layer of abstraction away from the underlying physical hardware. The techniques provide a complete virtual hardware environment in which a guest OS can execute in isolation and where resources are multiplexed transparently between concurrently executing OSs.

Full Virtualization (FV) involves the creation of hardware devices purely in software to provide an adequate supply of simulated hardware for a guest Operating System (OS) to run unmodified. This comes at a considerable performance penalty due to the interpretation of hardware operations in the guest [198]. Paravirtualization (PV) imitates a device interface using a far more direct path to handle devices inside a VM and can achieve better performance than FV. A downside of this technique is that it requires the installation of altered device drivers into a guest OS. A benefit is that this reduces the amount of time a guest spends accessing the device by relocating execution of critical tasks to its host where such tasks are more performant. HVM of a guest utilizes the additional hardware capabilities of an underlying host and provides the best performance of all the Virtualization techniques. Currently this takes the form of VMX within the instruction set of a host processor. This accelerates and isolates context switching between processes running in different VMs, increasing computational performance as instructions are directly passed to the host processor without having to be interpreted and isolated by the VMM. Unfortunately this technique comes at the expense of limiting the guest to using

the same instruction set as the host. Complete support for HVM of all computer subsystems, i.e. I/O peripherals, has yet to be fully realized in commodity computer hardware. However the performance benefits of HVM I/O have been explored using directed I/O Virtualization extentions [62].

5.3.2 XEN and KVM

Historically XEN has concentrated on the development of PV guests. Recently both XEN and KVM support multiple Virtualization techniques often referred to as Hybrid Virtualization [175]. Hybrid Virtualization combines the principles of both HVM and PV to obtain near native performance for guest OSs. This however comes attached with the disadvantages of both techniques, altered OS device drivers are necessary, along with modern VMX supporting hardware. XEN and KVM both contain built in Full Virtualization support, via the integration of QEMU [200], for OSs for which Paravirtualized device drivers do not exist or where hardware support for HVM is limited but comes at the cost of guest performance. QEMU is an open source FV VMM providing emulation of both IO devices, such as network interface cards and CPU architectures through binary translation.

Before the combination of multiple Virtualization techniques in KVM, the consolidation of an organization's current hardware using KVM was not feasible if its infrastructure did not support the instruction set extensions necessary for HVM guests. KVM conversely provides an excellent foundation for the creation of new virtual infrastructure through a reduction in the number of physical machines required for peak operating demand. This had the effect of reduced hardware running and setup costs. XEN on the other hand, with its better support for Paravirtualized guests, was more appropriate for deployment onto older hardware, providing acceptable performance as well as the benefits of consolidation.

Comparing XEN and KVM further, the lack of support for fully Paravirtualized guests in KVM across all OSs, such as the closed source and proprietary Microsoft Windows, has the potential to reduce performance. Alternatively the costs of porting Paravirtualized device drivers [21], to these OSs for XEN, do not exist. XEN is a more mature Virtualization solution and has been developed for longer than KVM pertaining to greater stability. However, KVM continues to be on the forefront of implementing new Virtualization techniques and is utilizing the latest research into HVM, providing greater performance improvements over the same implementations in software with the downside of requiring state-of-the-art hardware. One such technique introduced recently was hardware page-

table walkers that reduce memory address-translation overheads and remove the need for shadow page tables at the memory management unit level, providing memory savings and performance benefits [130] that were not initially available in XEN.

Another non-functional comparative advantage of KVM over XEN is that KVM is less pervasive due to its Hypervisor classification. As discussed in Chapter 2,XEN is classified as a Type 1 Hypervisor and KVM as a Type 2 Hypervisor according to [93]. Type 1 Hypervisors run directly on the host hardware with the guest OS running one level above the Hypervisor. In the case of XEN an installed micro-kernel is booted before the administrative guest "Domain0", which is used to configure other guests. On the other hand, Type 2 Hypervisors run from within a conventional OS, where the Hypervisor operates as a distinct second software layer and the guests execute above in a third layer. KVM is comprised of a single module probed into a standard Linux kernel. The comparative advantage of this is that a considerably smaller code-base has to be maintained, which lowers the risks of introducing bugs and reduces the amount of code to optimize.

5.3.3 Block I/O Paravirtualization

KVM and XEN have different PV architectures for accessing virtual block devices or virtual Hard Disk Drives (HDD) within a guest OS. One of the aims in this thesis is to evaluate the performance of these devices to ascertain the suitability of virtual infrastructure for data intensive applications, where maximising data throughput is critical.

KVM relies heavily on its integration with QEMU for a wide variety of storage backends. It has adopted Virtio [216] a Linux standard for virtual I/O device driver Application Binary Interfaces (ABI) where in essence the guest OS is aware of the virtual environment in which it is running, cooperating with the VMM, attaining higher performance and the benefits of PV. Virtio uses a layer of abstraction and a memory ring buffer as a transport for data between a VM and its host as expressed in Figure 5.3.

This provides the ability to write generic front-end virtual device drivers and arbitrary back-ends to support different device types for different OSs and VMMs. It removes the need to maintain multiple sets of virtual device drivers for each brand of VMM available. The XEN approach is very similar to that of KVM, with Virtio based considerably on the works of the XEN developers but does not rely solely on QEMU for the PV of devices. XEN supports block devices through a "hypercall" ABI that makes use of an altered version of the Linux `blkback` device, used for user land access to block devices, named `blktap` or "block tap", in combination with a frontend driver embedded in a guest. Paravirtualized support for non-modifiable OSs, such as Windows, has been implemented

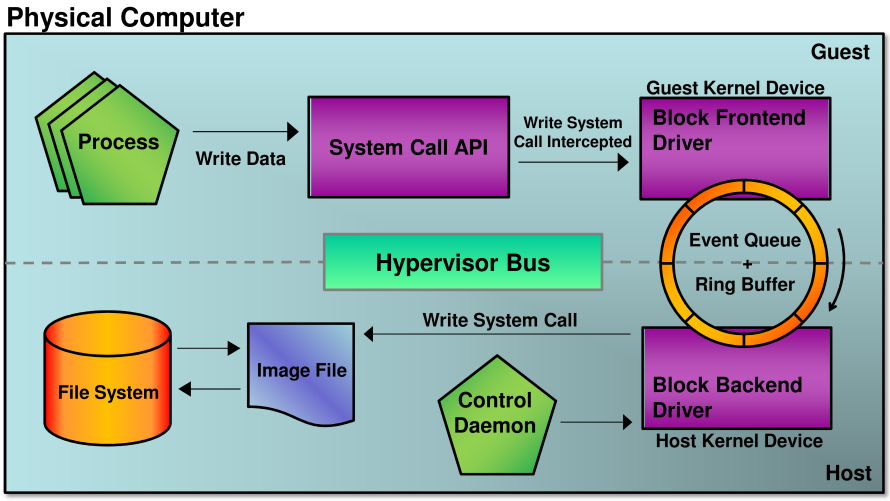


Figure 5.3. A simplification of virtual block I/O PV

in both KVM and XEN with emulated PCI devices replacing the ABI in traditional PV. The performance of these devices are planned to be investigated in future work. Section 5.4.4.3 discusses the implications of paravirtualization as part of an evaluation into the performance Block I/O devices at the Hypervisor level.

5.4 Evaluation

This section of the thesis introduces the testbed architecture and the benchmarks used to assess performance is, in addition to the experimental methodology followed. Finally, experimental results on image format, propagation delay and block I/O/ Paravirtualization performance issues are presented and analysed. The focus of the the evaluation is on components used at the infrastructure level of a Cloud.

5.4.1 Testbed Architecture

An experimental testbed was developed and used in the following experiments. At the time of experimentation it was comprised of four Dell commodity servers but more recently has been extended eight. Each server consists of a four core X3360 Intel Xeon CPU, running at the default clock speed of 2.83GHz and a total of 4GB of RAM (four modules of 1GB DDR2 at 800Mhz). Additionally, each server utilized a single 3.5 inch Western Digital RE3 250GB SATA HDD (Model: WD2502ABYS), with 16MB of cache and a spindle speed of 7200 RPM. The machines connect via Gigabit Ethernet using a Broadcom PCI-E NIC (Model: BCM95722). The file system in use on both the physical

machines and the VMs is EXT3. VMs were not provisioned concurrently and each instance used all available hardware with the exception of 2GB or half the available RAM of the host. The VM images are contained in raw format and stored on top of the host file system.

The OS present on both machines is Centos release 5.4 (Final). The following Hypervisor versions are used: KVM versions 83 and 2.6.32.24; XEN versions 3.4.3 and 4.0.1. Two Centos based Linux Kernel versions were used to test the older versions of KVM and XEN: 2.6.18-164.15.1.el5 for testing the performance of the native host and KVM version 83 guests; and 2.6.18-164.11.1.el5xen for testing the performance of XEN Dom0 and XEN 3.4.3 guests. The same vanilla Linux Kernel version 2.6.32.24 was used to test KVM version 2.6.32.24 and XEN version 4.0.1. Version 2.6 of Nimbus and version 2.0.1 of OpenNebula were deployed on the testbed. Globus Toolkit version 4.0.8 is installed enabling support for GSIFTP. The following versions of the benchmarking software, with justifications to follow in the next subsection, are used in the experiments: Bandwidth Monitor NG version 0.6 [36], IOzone version 347 [131] and Bonnie++ version 1.03e [28].

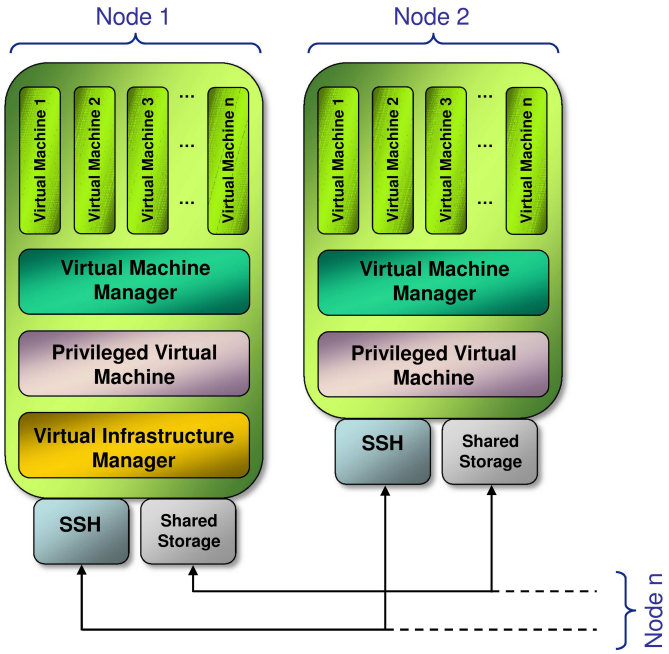


Figure 5.4. The testbed architecture used in the experiments.

Although the testbed is not a real representation of Cloud scale infrastructure, since it is comprised of only four machines, this resource based limitation should be considered acceptable for the purpose of the experiments. The primary interest of this research is to assess the overheads of the software components that comprise different elements of a Cloud system, either on a single physical machine or between two and four physical

machines. This is in contrast to assessing the scaling performance of a Cloud system in its entirety where a large number of physical resources would be necessary for any testbed to be representative of performance attainable in the real world. However if the need does arises in the future, the testbed architecture as outlined in Figure 5.4, was developed in such a manor as to be easily extended with additional physical machines.

5.4.2 Benchmarks

Three benchmarks partake in the performance evaluation: the first a novel and bespoke image propagation benchmark for reviewing the overheads of transferring a VM image between two nodes on a Cloud system; and two others that can be used to expose the performance of VMM block I/O systems.

The image propagation benchmark integrated Bandwidth Monitor NG (BWM-NG) as a tool to measure the input and output of the NIC from where the VM image repository and VIM were resident, referred to in this thesis as the head node. The benchmark involved the following steps:

i) Scripts to amalgamate the commands necessary for a user to provision a VM, inclusive of transferring image data to an appropriate host machine and then starting up the image as a VM instance

ii) Verification that a VM was on-line and executing via resource monitoring

iii) Termination of the VM in an automated fashion

For clarity, data was transferred across a homogeneous Cloud Environments that made use of the same Virtualization infrastructure. These steps facilitated the ease at which iterations of the experiment could be repeated. Each action was timed and compared to the start and end of the data transfer recorded by BWM-NG. This divulged the overheads of each stage of the image propagation, from the head node to the destination node where a VM was deployed. The image propagation benchmark takes a single parameter, the size of image to be transferred. The benchmark was implemented around three platforms: OpenNebula using the SCP protocol, Nimbus using the SCP protocol and finally Nimbus using GSIFTP.

Two synthetic benchmarks: IOzone and Bonnie++ were chosen for the performance evaluation of image formats and virtual block I/O devices. The motivation behind using these benchmarks over others was related to their design. These benchmarks were created to assess the performance of I/O devices only. Other benchmarks generally assess the

performance of CPUs devices and would generate useless metrics for any sensible comparison. These synthetic benchmarks try to encompass all possible parameters that could represent different workloads, such as database software writing small files randomly, to file servers reading large files sequentially. IOzone is a benchmark that generates and measures a selection of file operations to analyse the performance of a file system. Bonnie++ is a benchmarking suite with the ability to perform several tests to assess the performance of a file system and underlying HDD. In addition to monitoring the above Bonnie++ also monitors the CPU overhead of a given test to give a good indication of the efficiency of the block I/O implementation. The benchmarks were chosen because they perform very similar tests to record the number of I/O operations and the maximum throughput sustainable from a block device. This allows the results to be validated and any anomalies that differ between the benchmarks to be ruled out as implementation specific issues.

The following relevant IOzone tests, with accompanying definitions, were selected for the experiments:

- *Write* – Measures the performance of writing a new file, inclusive of writing file metadata.
- *Re-write* – Measures the performance of writing to a file that already exists, omitting much of the workload required in writing metadata.
- *Read* – Measures the performance of reading an existing file.
- *Re-read* – Measures the performance of reading a recently read file, illustrative of caching affects that can improve performance as reads are satisfied by cache.
- *Random Read* – Measures performance of reading random locations within a file, indicative of the performance of cache hierarchy and seek latency.
- *Random Write* – Measures performance of writing to random locations within a file, indicative of the performance of cache hierarchy and seek latency.

The parameters for the IOzone benchmark were left default with the exception of the test file size. From the IOzone documentation, this parameter needs to be double the available system RAM for results to record the performance of the underlying file system and block device as well as the caching effects of CPU and RAM. The automatic benchmark mode, shipped with IOzone, was used to perform a parameter sweep of the variables: file size used by the tests; and record size, the consecutive data chunk written to the file. The outcome of the parameter sweep is shown in Figure 5.5.

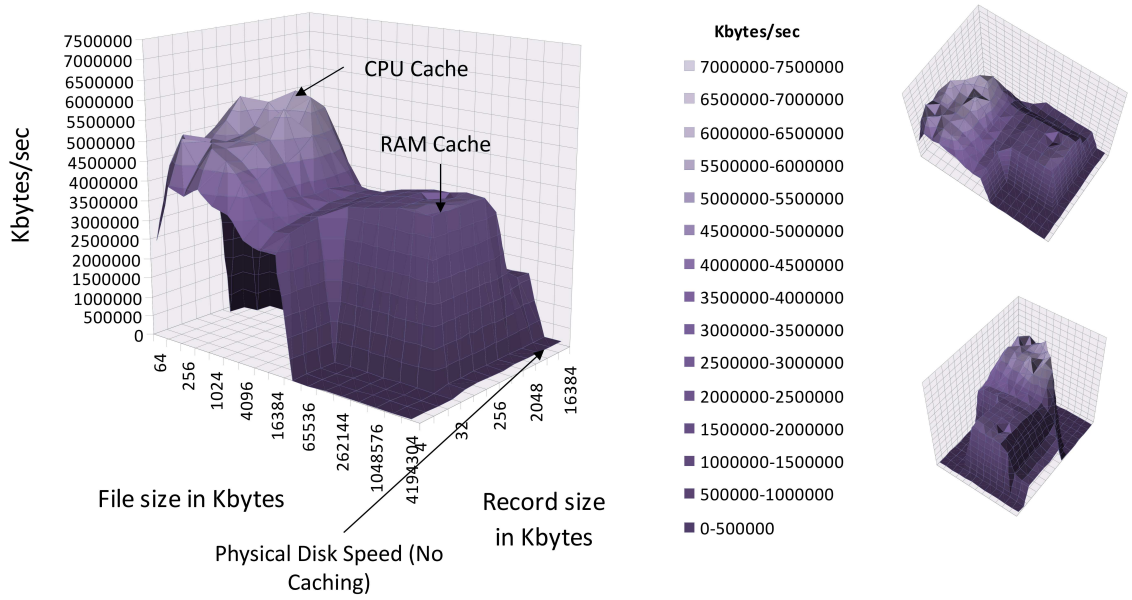


Figure 5.5. IOZone Parameter Sweep Caching Effects

The subsequent Bonnie++ tests with accompanying definitions were recorded during the experiment:

- *Sequential Throughput* – The number of blocks of data that can be read or written adjacent to one another in respect to the physical medium, in kilobytes per second.
- *File Operations* – The number of sequential or random, create or delete file operations per second illustrative of the overheads associated with manipulating file metadata in a file system.
- *Random Seek Operations* – The number of operations per second to seek to the beginning of a random file on a file system demonstrating the speed at which a HDD head can move from one physical track of a platter to another.
- *Read File Operations* – The number of sequential or random read operations of file metadata in zero size files per second an indicator of the efficiency of a file system's structure.

Default parameters were used for each iteration of Bonnie++ with the exception of the number of files to be created for the Read File Operations test which was set to "-n 512" or 524288 files. This enabled a reasonable degree of accuracy, where lower values would not be recorded because of safeguards in Bonnie++ that prevent high variance due to timer granularity. Bonnie++ correctly selected a test file size of twice the amount of

available RAM to prevent caching from effecting results and a fallacious representation of the underlying performance of the block I/O device.

5.4.3 Methodology

For clarity, this section of the thesis discusses the variables utilized in each of the experiments and discusses the selection of independent and dependent variables for each of the benchmarks. The image propagation experiment alters the image size, in the range 3GB to 21GB, in increments of 3GB and records time in seconds. It is difficult to gauge what sizes of image are representative of those used in the real world, as the image size is often dependent on the application but the chosen range should provide enough data points for any relation between the variables to be observed. For the IOzone and Bonnie++ experiments, the software stacks on which the benchmarks run on are altered. The software stacks are the native host, KVM Hypervisor deployed on the native host, XEN Domain0 or privileged guest that facilitates XEN VMs and XEN DomainU or XEN guest VM. In the image format experiment all working and available formats were benchmarked in KVM and XEN.

Depending on which benchmark test is performed, either the throughput in bytes per second or the number of operations per second is recorded. The percentage CPU time, where applicable in the case of Bonnie++ tests, is recorded in addition. Ten iterations of each of the benchmark tests are performed and the average result along with the standard deviation are presented, where possible, in illustrations.

5.4.4 Experimental Results

In this section, the results and interpretation of each of the three benchmarks are discussed, as outlined in the Section 5.4.2. Firstly, the results of the image formats are discussed. Second, the results of the image propagation benchmark are presented, trends are communicated and graphs are supplied that illustrate the differences between the overheads seen in each platform as image size is increased. Finally, the results of the two benchmarks used to ascertain the performance of the VMM Block I/O devices are analysed.

5.4.4.1 Image Format Performance Analysis

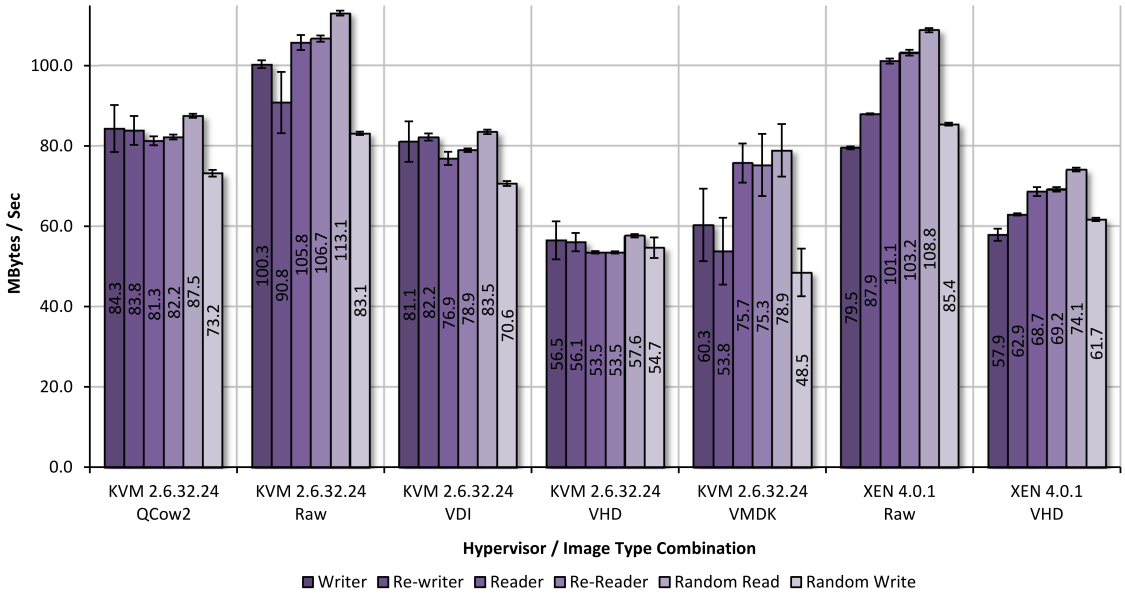


Figure 5.6. IOZone - Performance of Image Formats

In this section of the thesis the performance of all the supported image types for both the latest versions of Hypervisors, at the time of experimentation, are presented using the IOZone benchmark with a 16MB record file size and 4GB file size. In the case of KVM the cache is turned off as the results in Section 5.4.4.3 have shown that this provides the best performance (full results for all caching mechanisms can be found in Table 5.2). From Figure 5.6 it can be seen that performance differs depending on what image type is used. XEN and KVM with a raw image type are the most performant. KVM with QCow2 and VDI image types accomplish 80 to 85% the performance of the raw image type. Of all the images types tested KVM's VHD support shows the least promise. In comparison XEN's implementation of the VHD image type outperforms that of KVM's.

It is disappointing that native support for other image types in XEN is lacklustre at best and limited to the VHD image type. With the migration from `blktap` to `blktap2` support for QCow2 and VMDK is no longer available in XEN 4.0.1 `pv-ops` kernels. This limits the interoperability of XEN in comparison to KVM as unsupported image types would have to be converted.

5.4.4.2 Propagation Delay Performance Analysis

The results of the image propagation benchmark provide an interesting insight into how the underlying design and implementation of the image management supervision pro-

Table 5.2. Full Performance Results of Image Performance Analysis

Hypervisor	Image Format	Cache Type	Test Type					
			Writer	**Re-writer**	**Reader**	**Re-Reader**	**Random Read**	**Random Write**
KVM	**QCow2**	**None**	84.3MB/s (-/+ 5.8)	83.8MB/s (-/+ 3.6)	81.3MB/s (-/+ 1.1)	82.2MB/s (-/+ 0.6)	87.5MB/s (-/+ 0.5)	73.2MB/s (-/+ 0.8)
		Writeback	59.4MB/s (-/+ 6.6)	54.0MB/s (-/+ 2.5)	64.7MB/s (-/+ 2.3)	81.8MB/s (-/+ 0.7)	84.2MB/s (-/+ 0.7)	46.2MB/s (-/+ 1.6)
		Writethrough	49.3MB/s (-/+ 1.5)	43.7MB/s (-/+ 1.2)	79.0MB/s (-/+ 0.8)	80.4MB/s (-/+ 1.2)	83.2MB/s (-/+ 1.3)	32.4MB/s (-/+ 1.3)
	RAW	**None**	100.3MB/s (-/+ 1.0)	90.8MB/s (-/+ 7.6)	105.8MB/s (-/+ 1.9)	106.7MB/s (-/+ 0.8)	113.1MB/s (-/+ 0.6)	83.1MB/s (-/+ 0.4)
		Writeback	14.5MB/s (-/+ 1.4)	14.0MB/s (-/+ 0.5)	76.9MB/s (-/+ 1.2)	78.7MB/s (-/+ 1.3)	84.3MB/s (-/+ 0.7)	13.2MB/s (-/+ 0.4)
		Writethrough	61.3MB/s (-/+ 4.5)	55.5MB/s (-/+ 2.1)	65.8MB/s (-/+ 3.9)	86.6MB/s (-/+ 0.6)	88.5MB/s (-/+ 0.6)	49.6MB/s (-/+ 1.7)
	VDI	**None**	81.1MB/s (-/+ 5.0)	82.2MB/s (-/+ 0.9)	76.9MB/s (-/+ 1.7)	78.9MB/s (-/+ 0.4)	83.5MB/s (-/+ 0.5)	70.6MB/s (-/+ 0.6)
		Writeback	57.9MB/s (-/+ 5.7)	51.6MB/s (-/+ 3.3)	64.1MB/s (-/+ 3.6)	78.8MB/s (-/+ 0.5)	81.1MB/s (-/+ 0.7)	46.0MB/s (-/+ 2.8)
		Writethrough	47.9MB/s (-/+ 0.9)	41.5MB/s (-/+ 2.1)	76.4MB/s (-/+ 1.4)	77.5MB/s (-/+ 0.8)	79.9MB/s (-/+ 1.0)	29.6MB/s (-/+ 9.3)
	VHD	**None**	56.5MB/s (-/+ 4.7)	56.1MB/s (-/+ 2.3)	53.5MB/s (-/+ 0.3)	53.5MB/s (-/+ 0.3)	57.6MB/s (-/+ 0.4)	54.7MB/s (-/+ 2.6)
		Writeback	32.5MB/s (-/+ 2.4)	38.6MB/s (-/+ 1.3)	54.6MB/s (-/+ 0.4)	54.5MB/s (-/+ 0.2)	59.1MB/s (-/+ 0.5)	23.3MB/s (-/+ 1.5)
		Writethrough	19.8MB/s (-/+ 2.2)	26.0MB/s (-/+ 1.5)	53.4MB/s (-/+ 0.9)	53.3MB/s (-/+ 1.3)	56.7MB/s (-/+ 2.2)	15.9MB/s (-/+ 0.4)
	VMDK	**None**	60.3MB/s (-/+ 9.0)	53.8MB/s (-/+ 8.3)	75.7MB/s (-/+ 4.9)	75.3MB/s (-/+ 7.7)	78.9MB/s (-/+ 6.5)	48.5MB/s (-/+ 5.9)
		Writeback	39.9MB/s (-/+ 3.8)	43.3MB/s (-/+ 3.1)	65.5MB/s (-/+ 4.5)	77.6MB/s (-/+ 0.6)	80.9MB/s (-/+ 2.8)	36.6MB/s (-/+ 1.4)
		Writethrough	20.4MB/s (-/+ 1.2)	26.9MB/s (-/+ 1.3)	77.5MB/s (-/+ 1.1)	77.5MB/s (-/+ 1.3)	82.2MB/s (-/+ 1.5)	14.6MB/s (-/+ 0.7)
XEN	**Raw**	**None**	79.5MB/s (-/+ 0.4)	87.9MB/s (-/+ 0.2)	101.1MB/s (-/+ 0.6)	103.2MB/s (-/+ 0.7)	108.8MB/s (-/+ 0.5)	85.4MB/s (-/+ 0.4)
	VHD	**None**	57.9MB/s (-/+ 1.5)	62.9MB/s (-/+ 0.3)	68.7MB/s (-/+ 1.1)	69.2MB/s (-/+ 0.6)	74.1MB/s (-/+ 0.5)	61.7MB/s (-/+ 0.4)

cesses, used in each IaaS manager, have an impact on the time it takes for a single iteration of a VM's lifecycle. Figure 5.7 presents these times against an increasing VM image size.

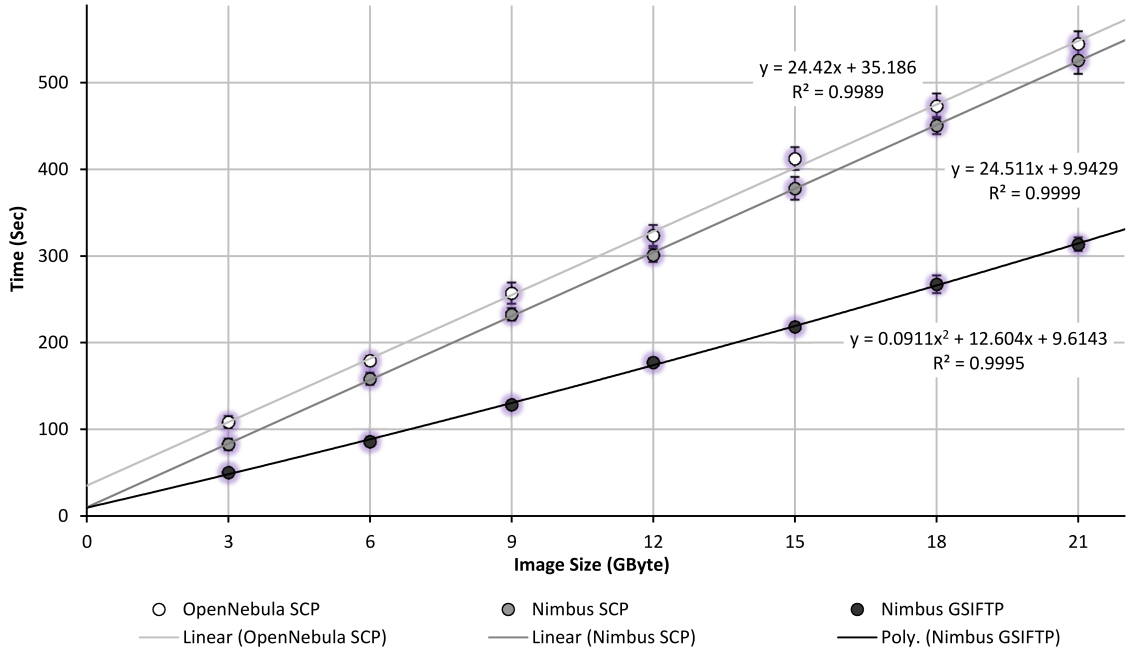


Figure 5.7. Image propagation: Trends

It is clear that Nimbus using GSIFTP as a transfer protocol out performs both Nimbus and OpenNebula using SCP. Adding appropriate trend lines, interpolating to an image size of zero and reading the appropriate value from the y-axis intercept, provides an indication of the overhead induced by the implementation within each IaaS manager, responsible for managing the propagation process of a VM. Figure reffig:ProtocalTransferTime illustrates the theoretical transfer time of the NIC with neither protocol obtaining this performance, demonstrates that on average GSIFTP transfers data nearly twice as fast or in $\sim$46% less time than SCP and when using the SCP protocol the transfer time is agnostic of the IaaS manager used.

Figures 5.9, 5.10, 5.11, 5.12, 5.13 and 5.14 show the results of the time spent waiting to change from one state to another in each of the platforms tested. It can be seen that the majority of the time is spent in a transferring state, waiting for the transfer protocol to complete the movement of data from the image repository to the VM host machine. On the other hand when transferring smaller images, the initial and terminal overheads of the IaaS manager supervising the propagation process, became an increased proportion of the total time spent waiting.

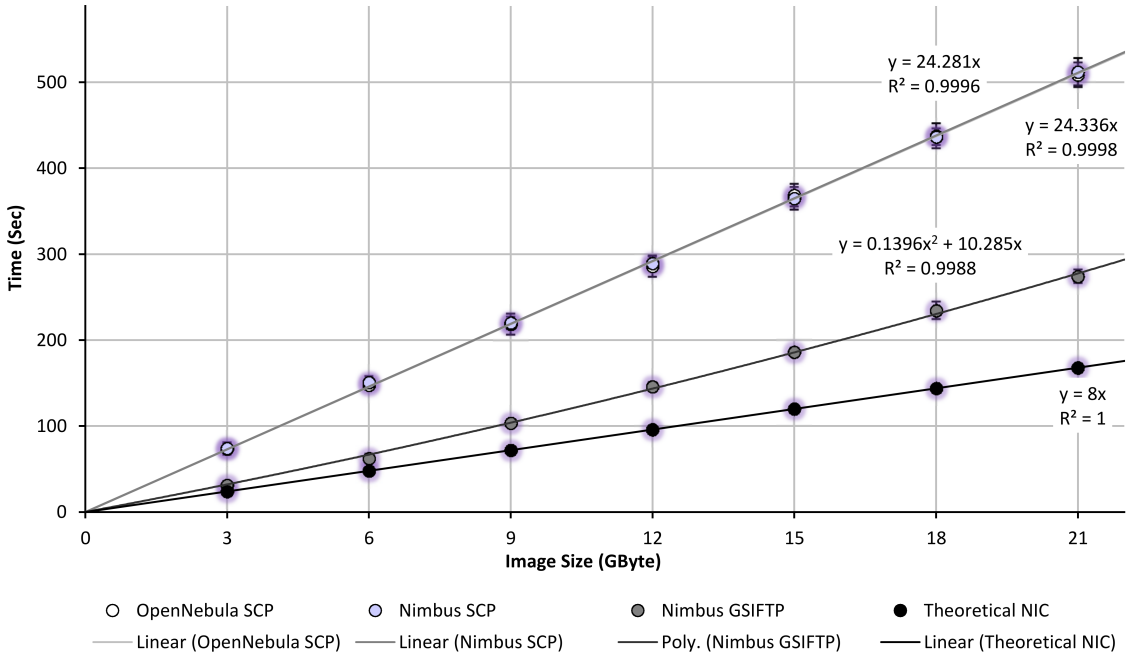


Figure 5.8. Image propagation: Protocol Transfer Time

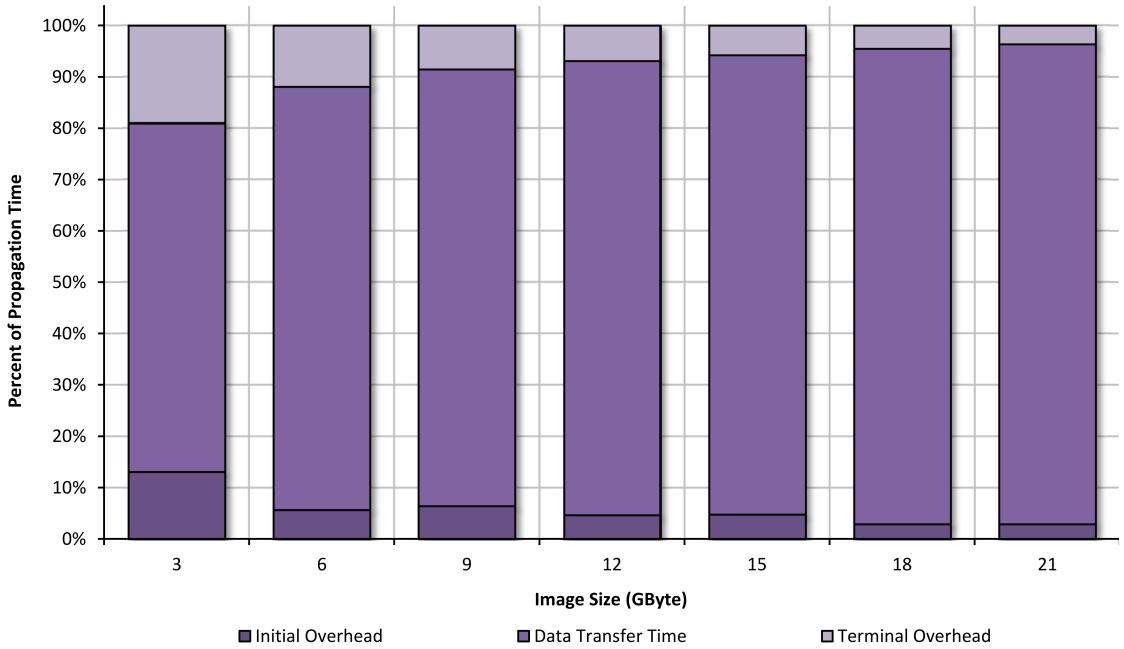


Figure 5.9. Image Propagation: OpenNebula SCP Overheads As A Percentage

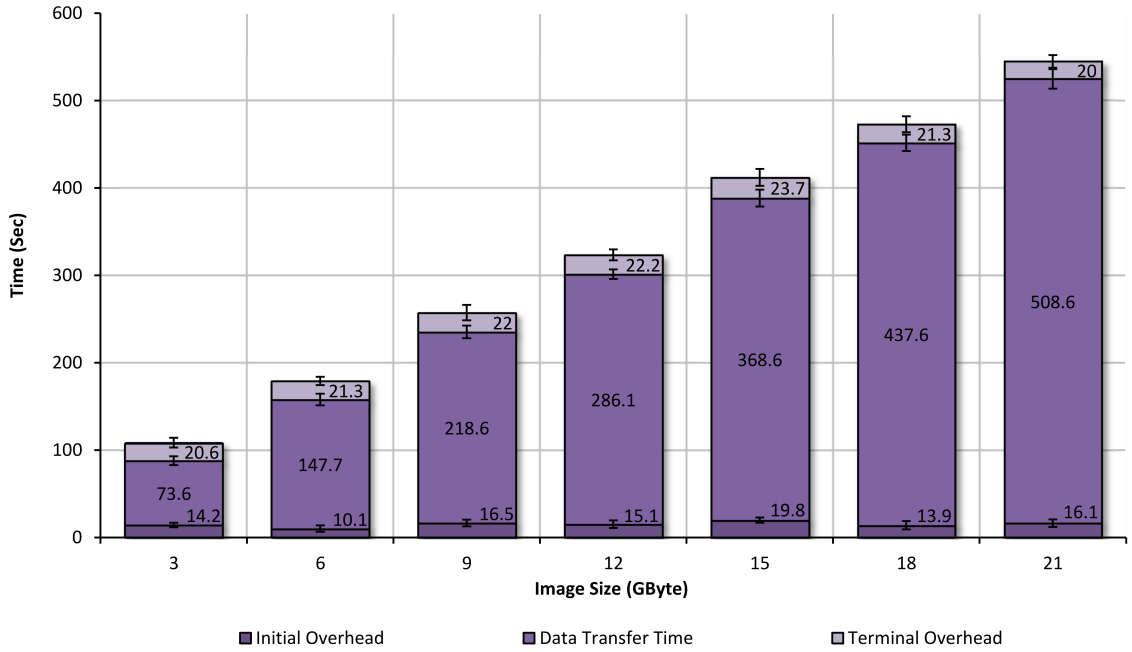


Figure 5.10. Image Propagation: OpenNebula SCP Overheads In Time

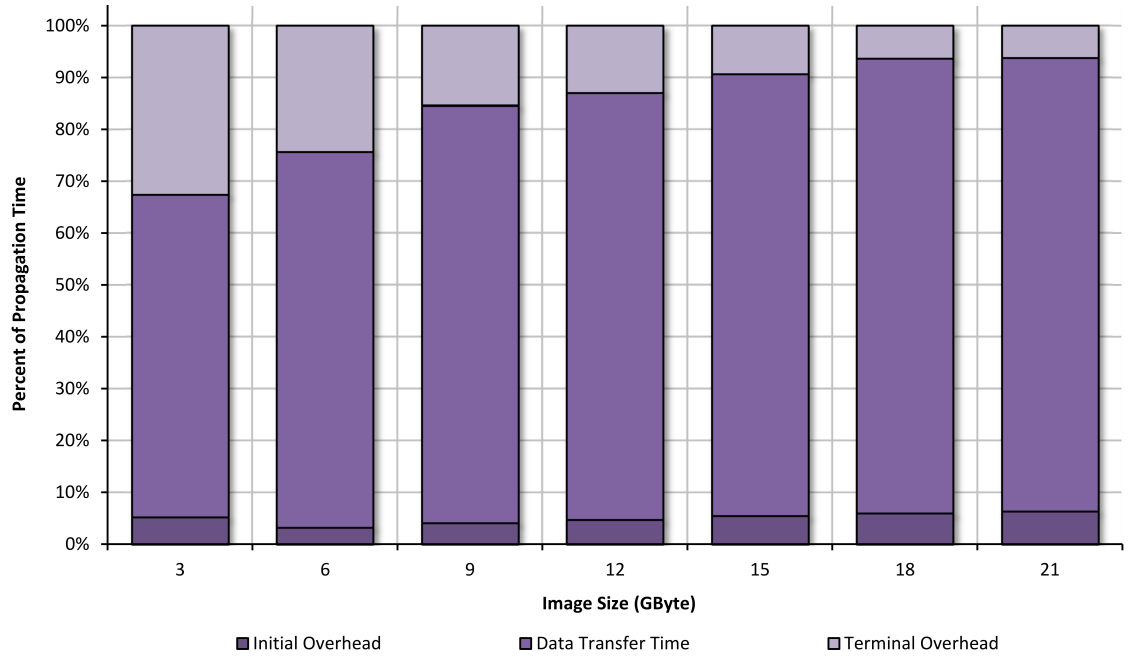


Figure 5.11. Image Propagation: Nimbus SCP Overheads As A Percentage

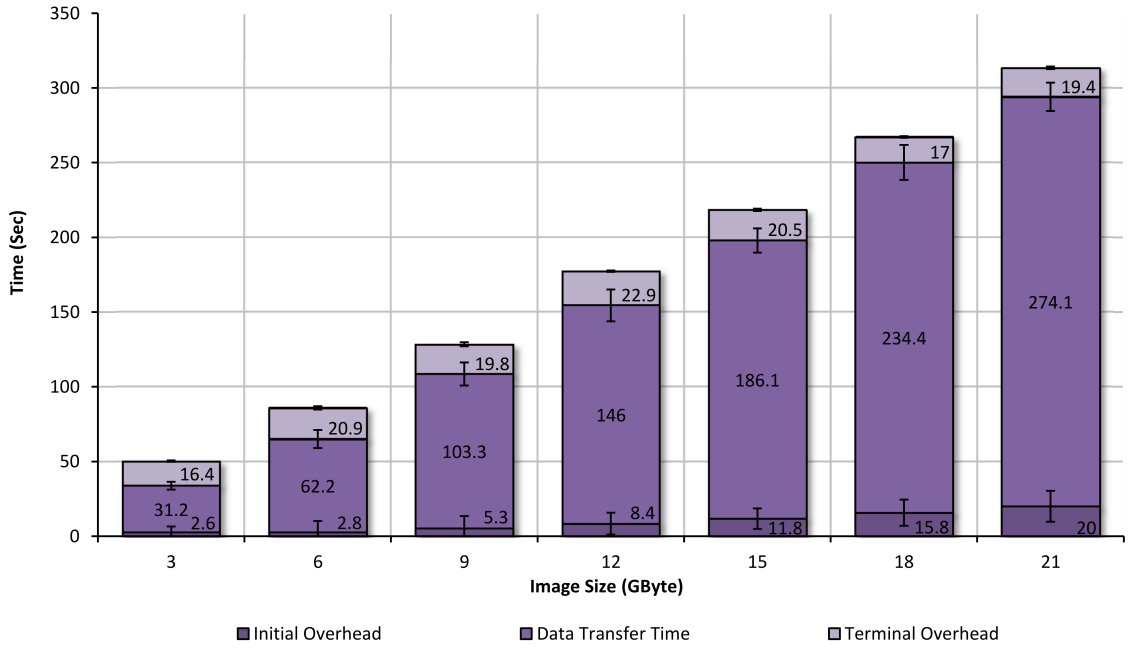


Figure 5.12. Image Propagation: Nimbus SCP Overheads In Time

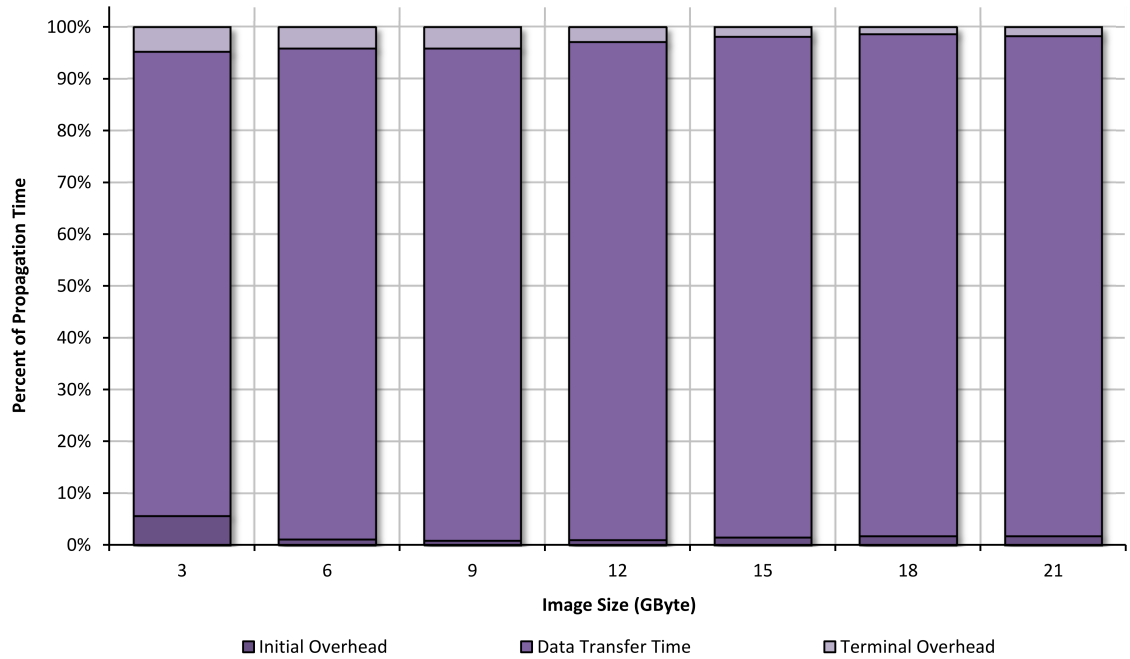


Figure 5.13. Image Propagation: Nimbus GSIFTP Overheads As A Percentage

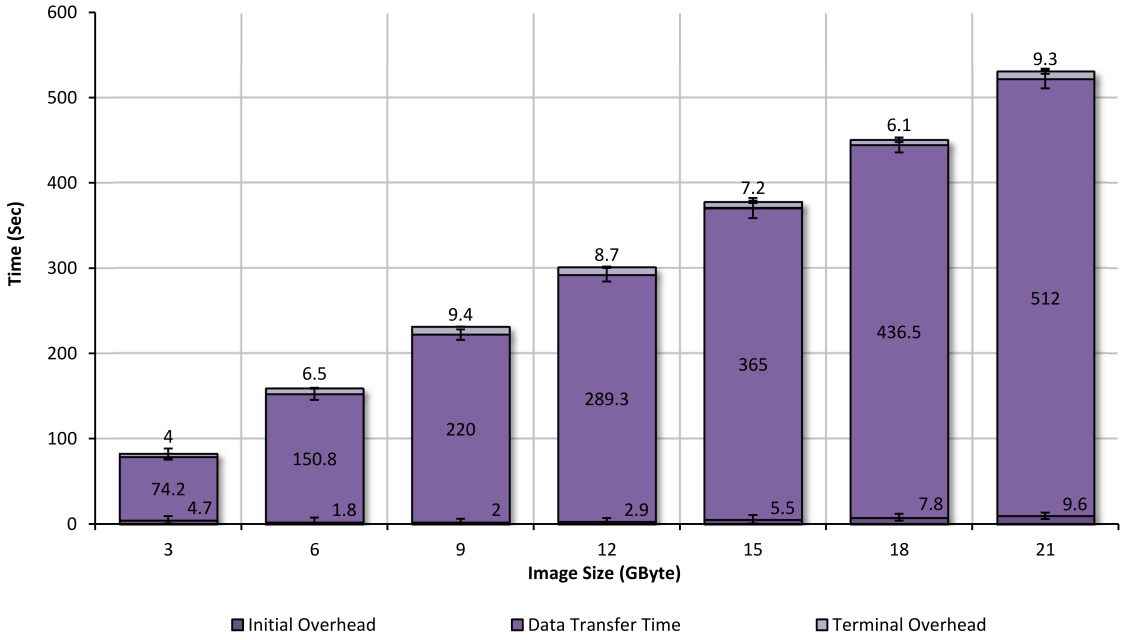


Figure 5.14. Image Propagation: Nimbus GSIFTP Overheads In Time

When comparing the SCP transfer protocol on different IaaS managers (Figures 5.9, 5.10 and 5.11, 5.12) Nimbus can be seen to outperform OpenNebula with a considerable reduction in the time spent in the initialization and termination states, inducing less overhead for identical sizes of image, specifically $\sim$3.5 times faster or in $\sim$72% less time.

When Nimbus utilizes the GSIFTP transfer protocol (Figures 5.13 and 5.14), the initialization time remains roughly proportional to the overall propagation time. This is in contrast to the other platforms where the percentage of time spent in the initialization and termination states decrease proportionally with an increase in image size, the overheads remain constant. This feature is also illustrated by the polynomial trend line in Figure 5.7.

An additional two servers were added to the testbed architecture to provide an insight into and assess the performance of OpenNebula to provision multiple virtual resources concurrently. The image propagation benchmark, as described in Section 5.4.2, was extended to enable the submission of multiple images. An additional step was added that forked a separate script for the submission and termination of a single resource. A parent script waited for all resource request cycles to end before terminating the benchmark. Six requests were submitted to three hosts, two VMs for each host and the VIM was given contention free access to the resources of the forth host to propagate the images using SCP. The overhead and transfer time of the multiple image experiments are illustrated in Figures 5.15 and 5.16.

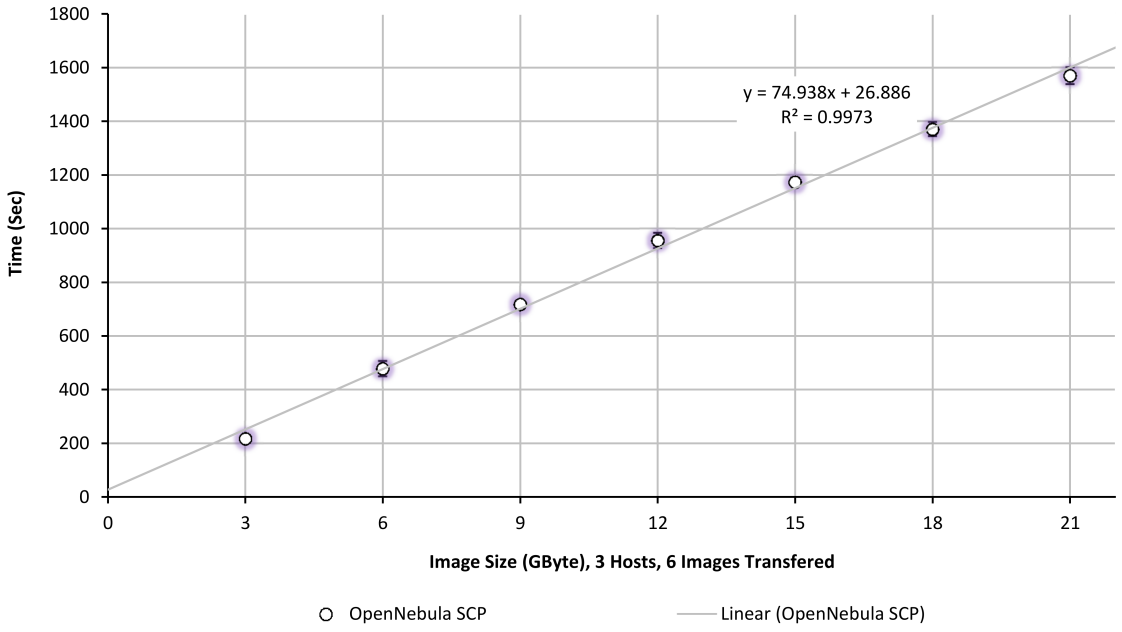


Figure 5.15. OpenNebula multiple image propagation: Trend

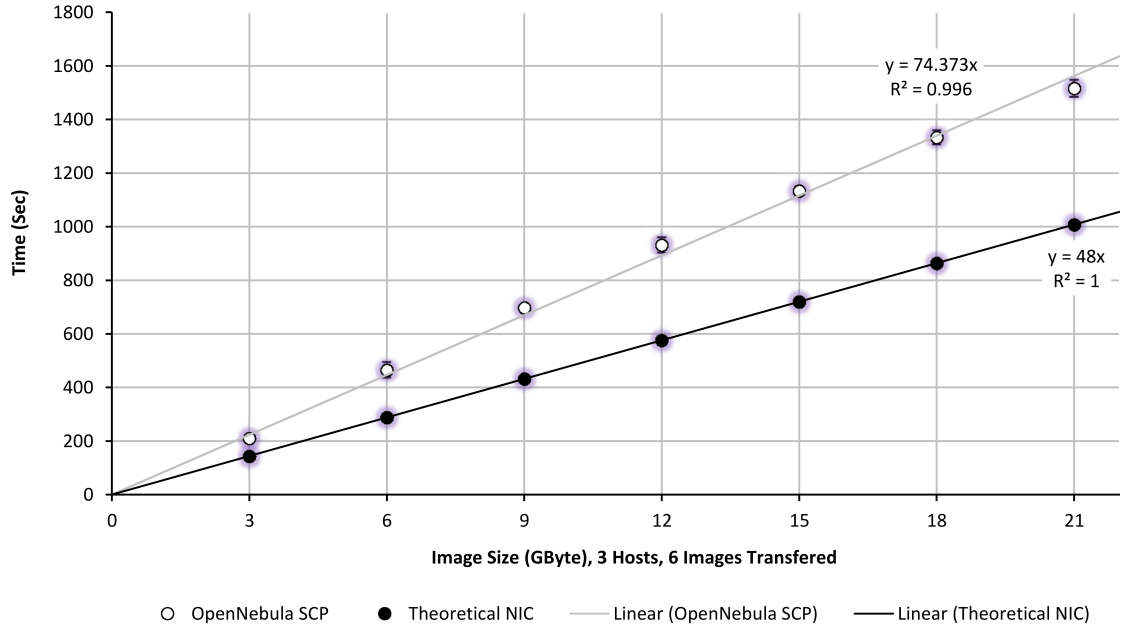


Figure 5.16. OpenNebula multiple image propagation: Protocol transfer time

It can be seen in Figure 5.15 that the overhead of provisioning multiple resources is nearly identical, to that of Figure 5.7, when variance is accounted for in comparison to that of a single resource. An interesting feature of Figure 5.16 shows that although six times as much data was transferred, compared to the single image experiment in 5.8, the

time to propagate all the images took roughly only three times as long. This demonstrates that the VIM was able to take greater advantage of the NIC, closer to that of the theoretical maximum limit, when executing concurrent transfers. The single image experiments only attained ∼33% utilization of the maximum throughput compared to ∼65% of the multiple image experiments. For clarity, the results presented in these figures are across ten experimental iterations and have minimal variance, barely visible in the graphs as error bars.

5.4.4.3 Block I/O Performance Analysis

Block I/O devices are incredibly slow in comparison to the performance of memory and CPU caches in traditional computer systems, with many millions of CPU cycles being wasted to service a single I/O requests when a cache miss occurs. Optimization of the virtual equivalent block I/O, like any leading bottleneck within a system, should be of a high priority and has repercussions for applications that utilize large datasets. The results in this subsection on the benchmarks of IOzone and Bonnie++, divulge the performance of the VMM block I/O devices and thus reveal the state of development, the amount of effort and time assigned to optimization and an indication of the maturity of the Virtualization solutions: KVM and XEN.

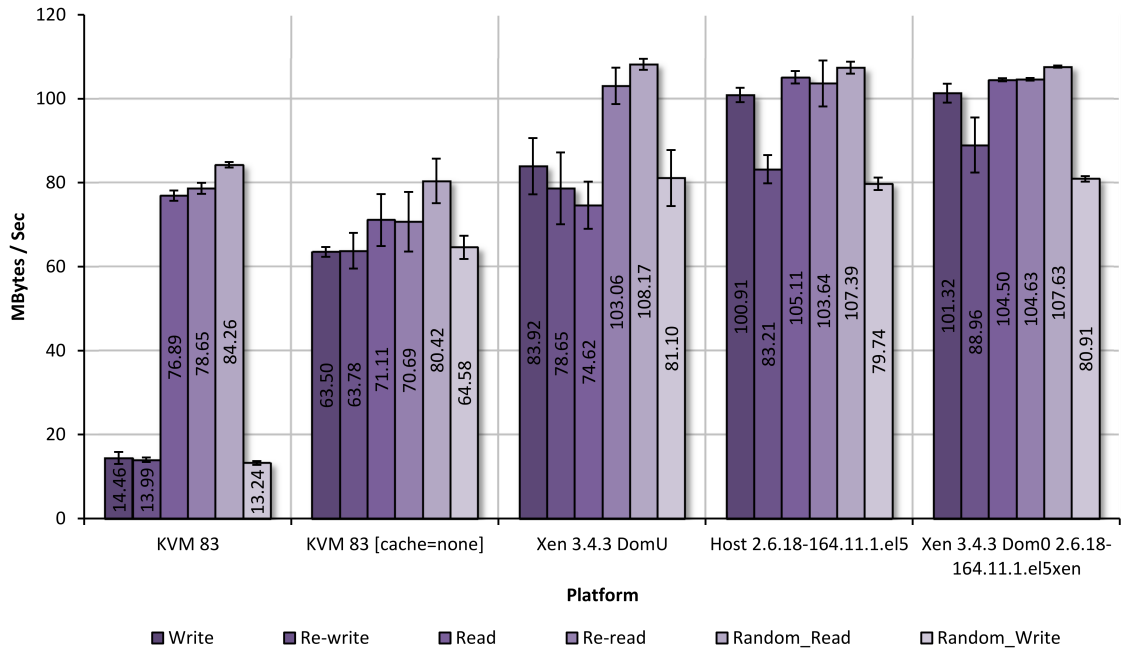


Figure 5.17. IOzone - record size: 16MB, file: 4GB guest; 8GB host

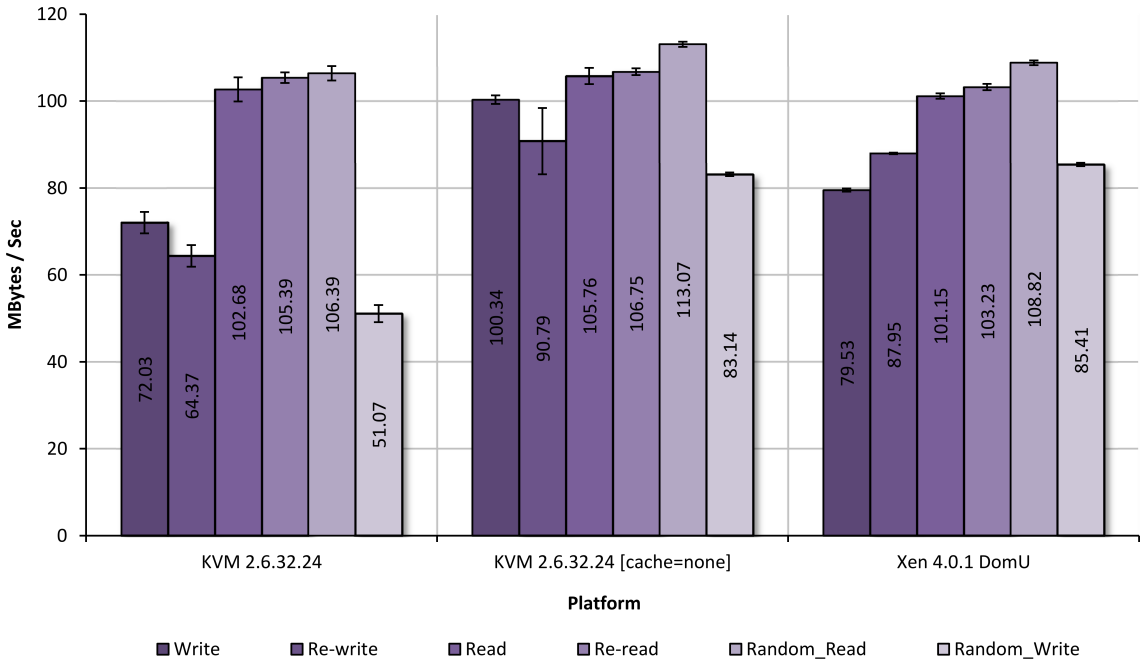


Figure 5.18. IOzone on 2.6.32.24 - record size: 16MB, file: 4GB guest.

Figure 5.17 and 5.18 illustrates the maximum throughput of the virtual block I/O devices for a given platform, where a record size of 16Mbyte is used to minimize CPU usage. It can be seen that out of the box, KVM 83 write performance is incredibly poor in contrast to XEN 3.4.3. KVM 83 on average across all write tests exhibited ∼17% of the throughput of XEN 3.4.3. IOzone read tests demonstrate that KVM 83 performs similarly to XEN 3.4.3 in the initial read test but failed to equal the throughput for the re-read and random read tests attaining ∼77% of the throughput. The tests on the newer versions of KVM and XEN are shown in Figure 5.18. KVM 2.6.32.24 again shows poor write performance but is a vast improvement over the older version by roughly a factor of 3. The performance of XEN 4.0.1 is on par with the older version of XEN 3.4.3.

After further investigation, the bottleneck was tracked to the caching system of the QEMU back-end used by KVM. By default a write-through cache is deployed for consistency to guarantee storage is committed to the underlying physical device. This has the effect of increasing the amount of bus traffic and additional copy operations needed and consequently reduces the performance of write operations. With this in mind, the benchmarks for KVM were rerun avoiding the use of the cache all together and this demonstrated far superior performance. This time KVM displayed ∼79% of the throughput of XEN. Figure 5.18 shows the new version of KVM 2.6.32.24 performing on par with and on occasion better than XEN 4.0.1.

Comparing XEN 3.4.3 with the native host platform and accounting for variance, write

throughput was on par on all tests other than the initial write test which revealed that XEN 3.4.3 exhibited ∼83% of the throughput. The results of the XEN 3.4.3 initial read test demonstrated similar results with ∼71% of the throughput of the host. The other XEN 3.4.3 read tests were indistinguishable from the host and the performance of the privileged guest "Domain0" was equivalent to the native host performance across all tests.

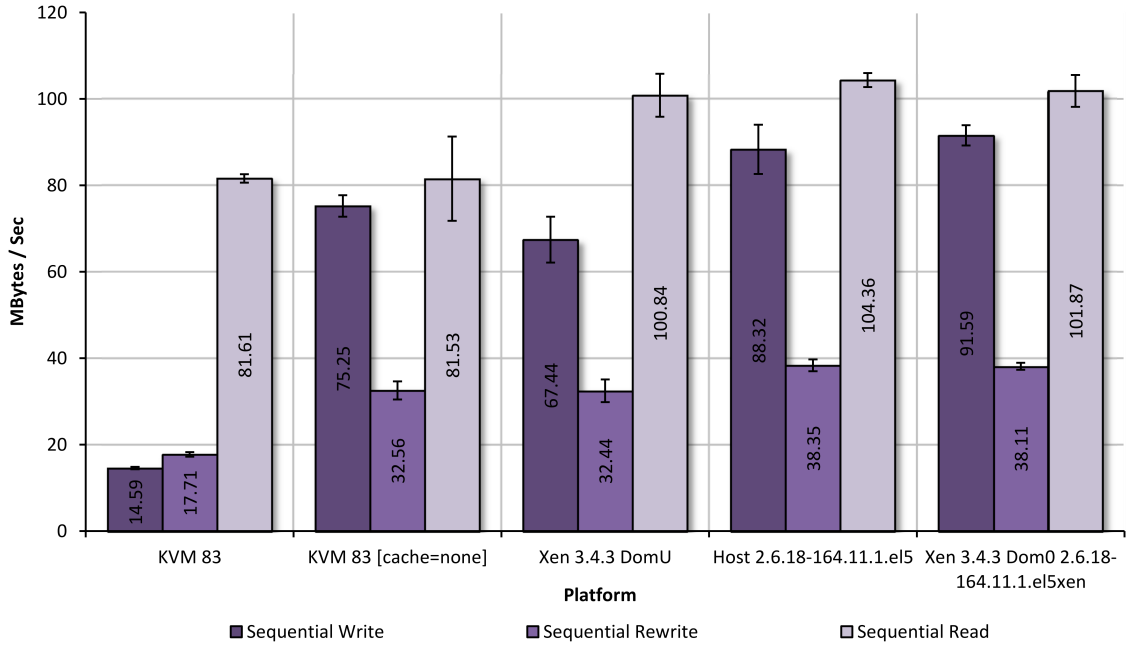


Figure 5.19. Bonnie++ - Throughput MB/Sec

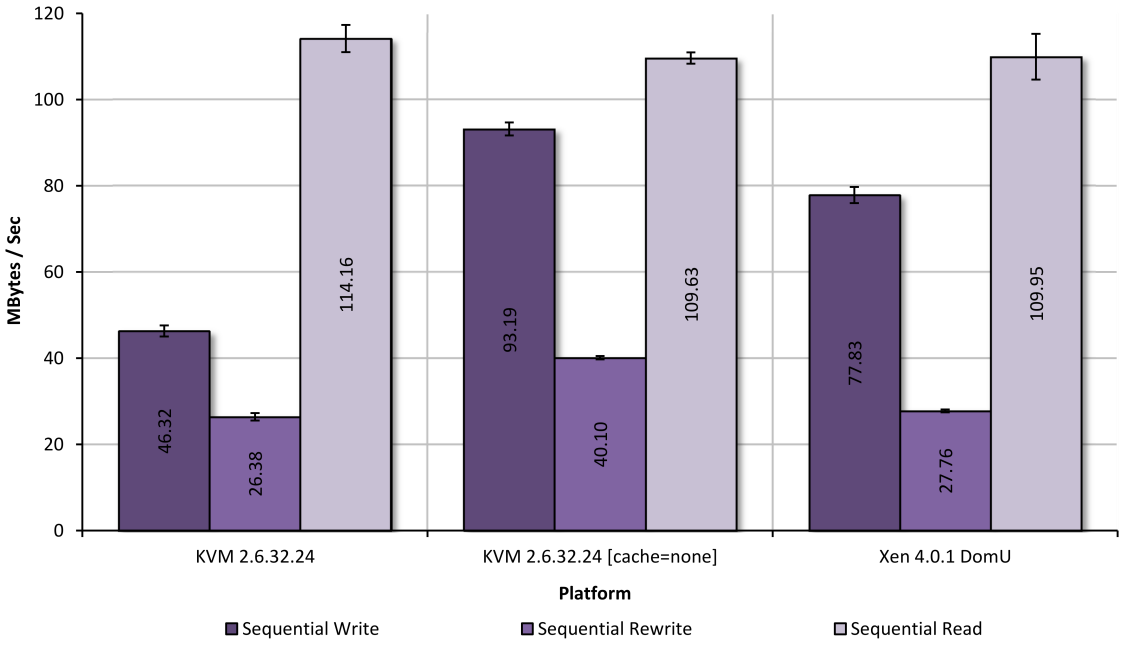


Figure 5.20. Bonnie++ on 2.6.32.24 - Throughput MB/Sec.

The results of the Bonnie++ benchmark for sequential throughput (Figure 5.19 and 5.20) are confirmatory, showing near identical results to IOzone for the sequential write and read tests but oddly show differing results for the sequential re-write test, across all platforms. Speculating on the cause of this discrepancy, the manner in which the re-write test is implemented may be the root cause. The results are roughly a 50% reduction in throughput compared to IOzone; excluding the already limited performance of KVM with cache. This could indicate that the sequential re-write test throughput is a formulation of the time to execute two consecutive sequential write operations and that this doubling of data written may not have been accounted for. This highlights the need for additional confirmatory benchmarks when running performance evaluation experiments, as the underlying implementation of a benchmark may not provide a fair representation of performance. Additionally, from the figures it can be seen that Xen Dom0 performs marginally better in some tests than the underlying host and is most likely due to the variance in the experimental results over multiple iterations.

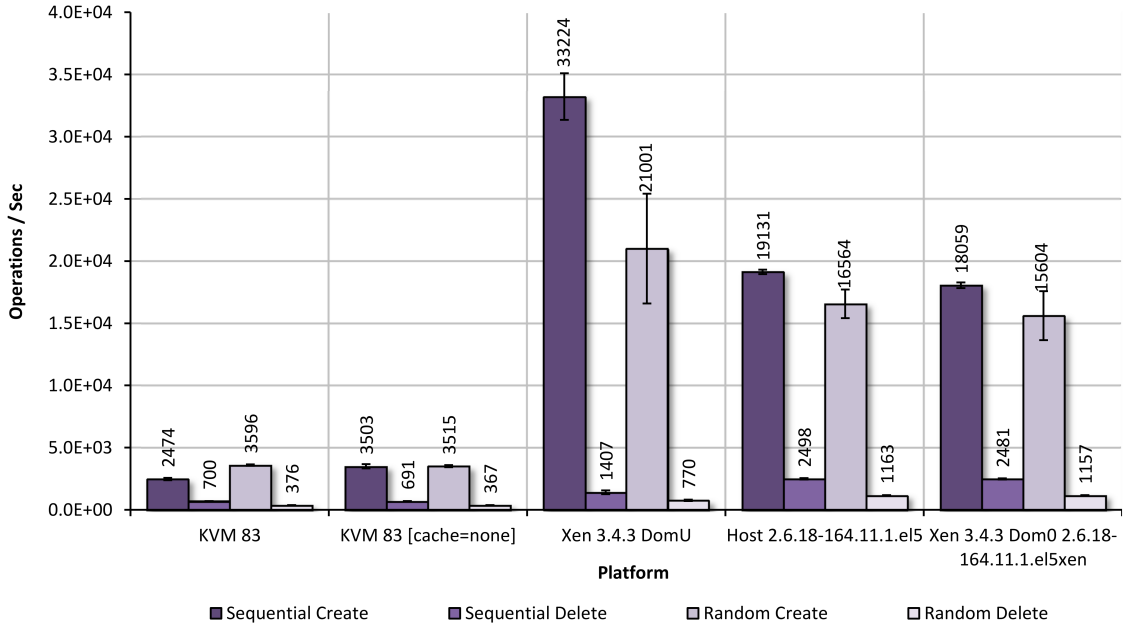


Figure 5.21. Bonnie++ - File operations per second

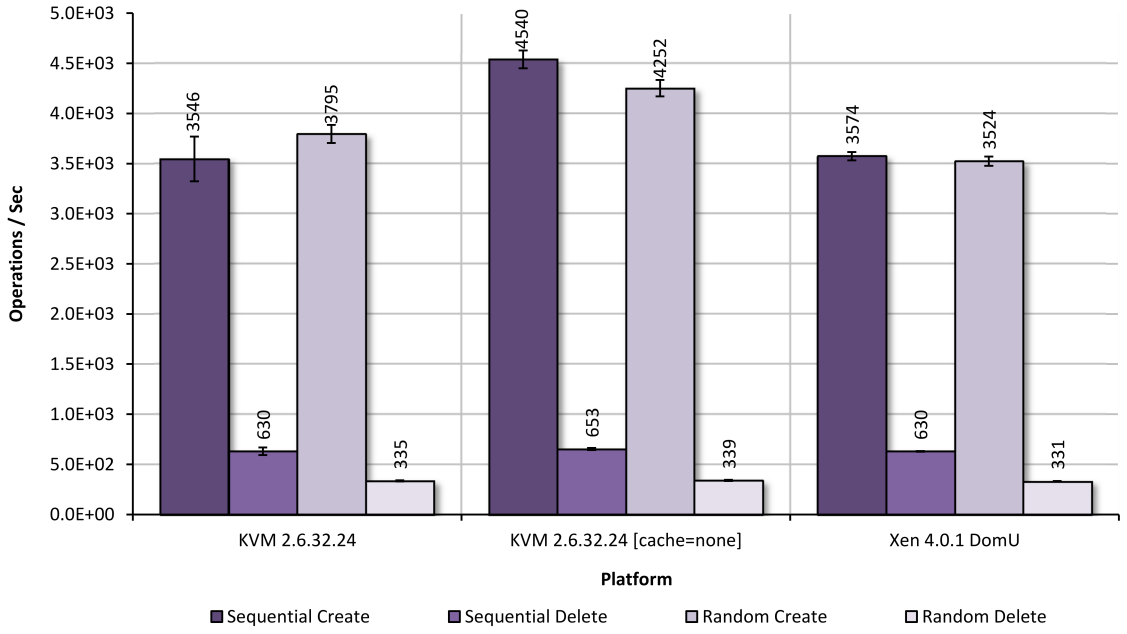
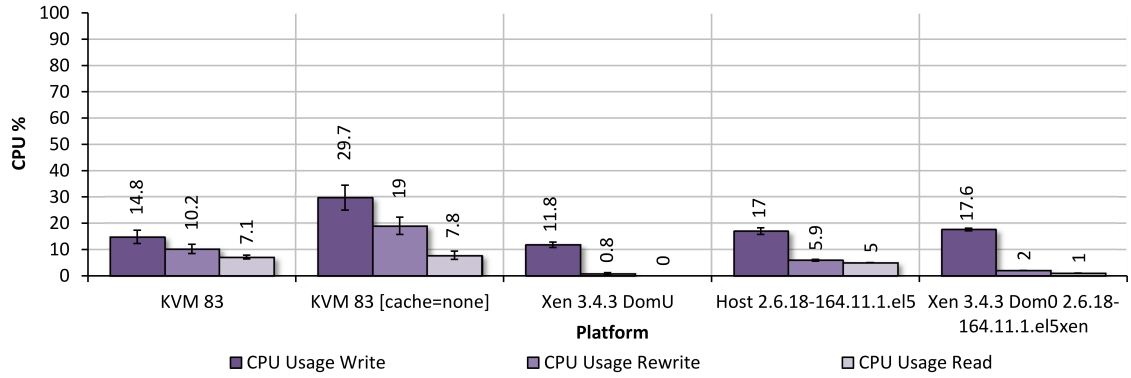


Figure 5.22. Bonnie++ on 2.6.32.24 - File operations per second.

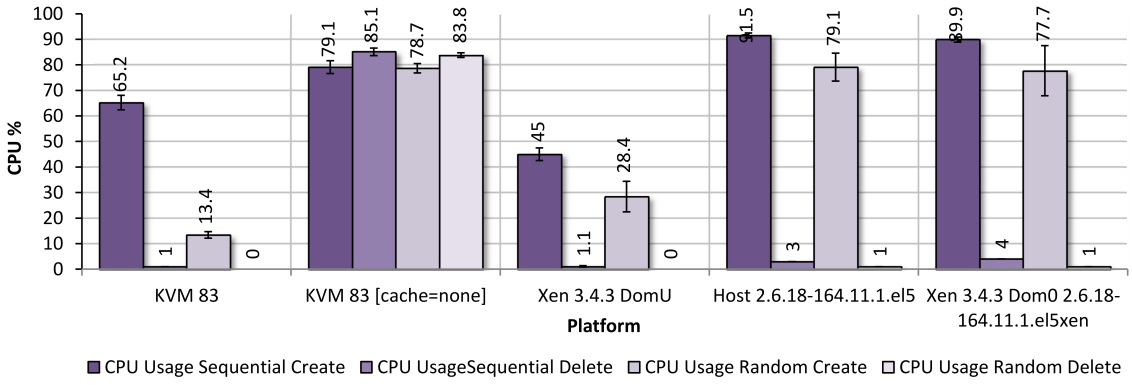
While throughput of the virtual block devices provides an indication of performance, the number of operations that can be performed per second give further insight into how efficient the implementations are. Figure 5.21 and 5.22 presents the Bonnie++ benchmark results for sequential and random deletion and creation of files respectively. The results

show that KVM 83 performs poorly in all cases, more so in the create tests where an order of magnitude less operations can be executed and where turning off the storage cache in KVM has minimal impact on performance. Interestingly XEN 3.4.3 guests outperform the native host possibly due to some undocumented caching. Again it can be seen that the performance of KVM 2.6.32.24 is comparable to XEN 3.4.3 and 4.0.1.

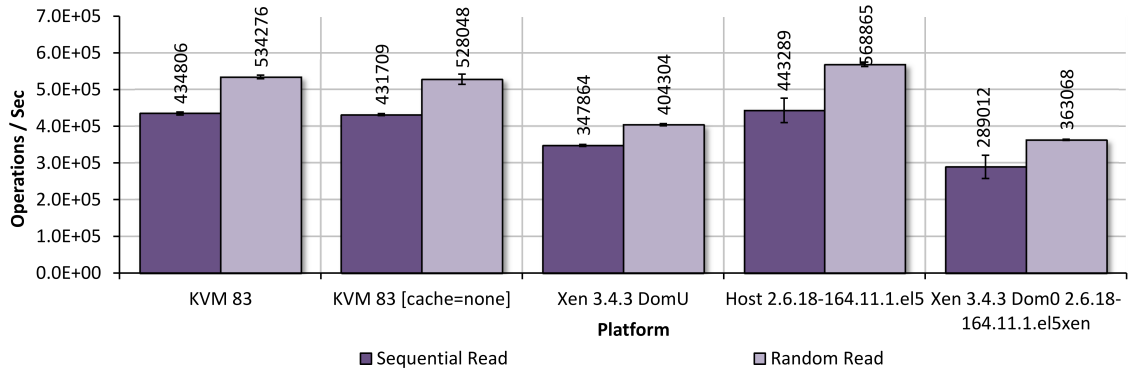
Another indicator of efficient design and implementation is that of CPU usage used when performing I/O related tasks. Applications using CPU resources can become starved by OS subsystems such as block I/O back-ends using excessive CPU time to service requests for accessing I/O devices. Figure 5.23(a) shows KVM 83 performing poorly on all sequential operations using a greater percentage of CPU to service less requests in comparison to XEN. Interestingly the XEN 3.4.3 guest performs better than the underlying XEN Domain0, exhibiting very little CPU usage if the CPU metric gathered within the XEN guest is to be trusted. Disabling the storage cache of KVM 83 creates an increase in the percentage of CPU time used. In the file operation tests of Figure 5.23(b) this performance gap is even more prominent with all tests exhibiting CPU usage around 80%. At first glance KVM with storage cache seems to outperform XEN in the percentage of CPU time used to create files randomly but performs far less operations per second. Figure 5.24(a) demonstrates another performance improvement for KVM 2.6.32.24 over KVM 83 and a slight performance regression from XEN 3.4.3 to 4.0.1. Figure 5.24(b) reveals a large reduction in CPU usage across the tests for KVM 2.6.32.24 with and without cache and a modest improvement for XEN 4.0.1 over XEN 3.4.3. For clarity, the 0% CPU cost displayed in some of the figures, is due to the course granularity of CPU measure used and is not in fact zero but a very small value that cannot be recorded.



(a)

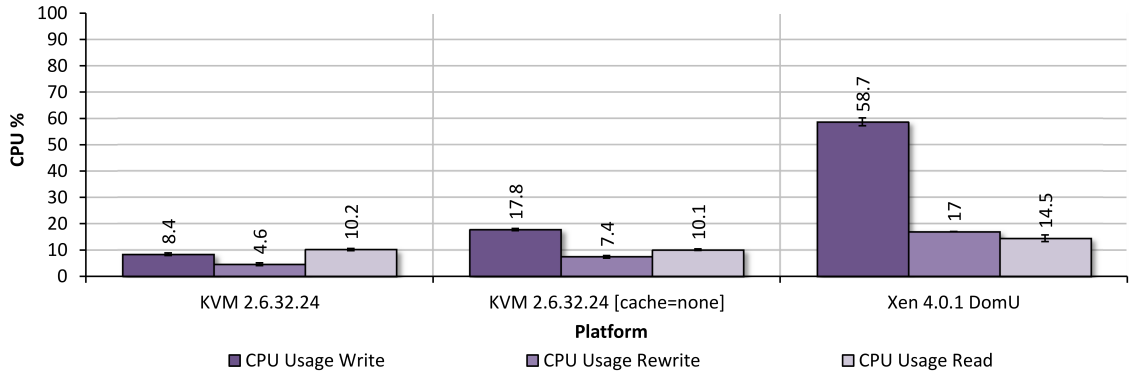


(b)

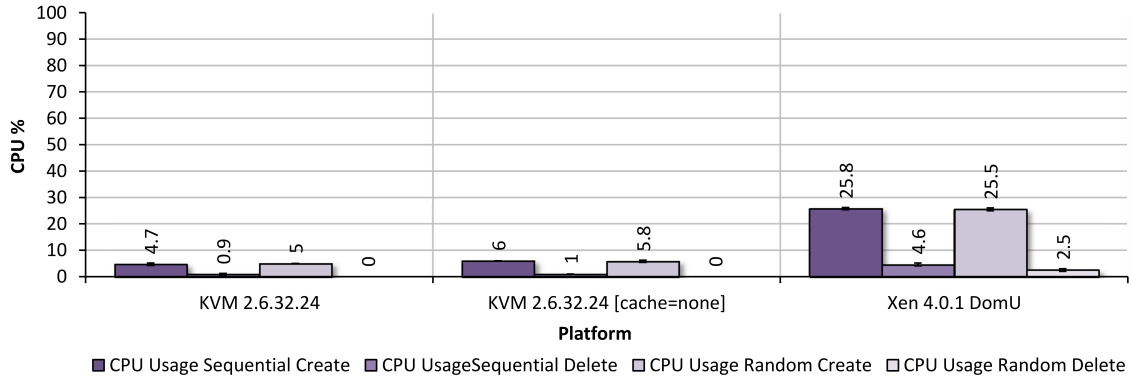


(c)

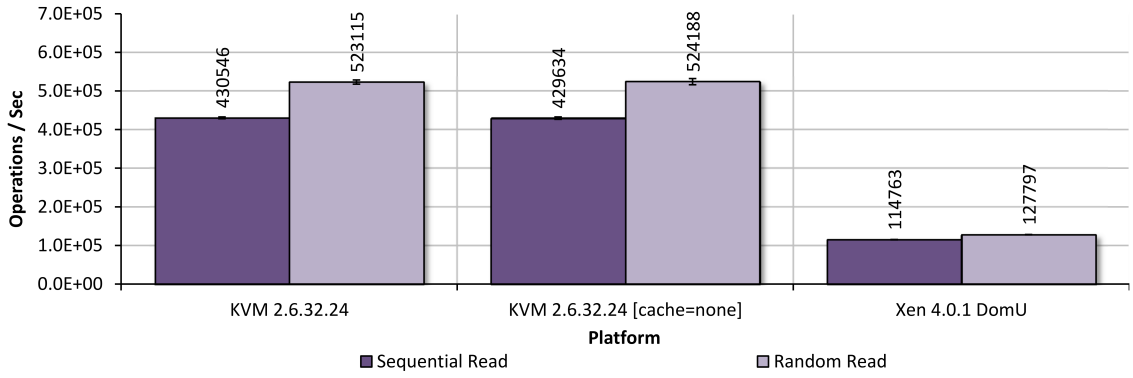
Figure 5.23. Bonnie++ - Top-to-Bottom: Sequential Throughput CPU Usage, File Operations CPU Usage & Read File Operations.



(a)



(b)



(c)

Figure 5.24. Bonnie++ on 2.6.32.24 - Top-to-Bottom: Sequential Throughput CPU Usage, File Operations CPU Usage & Read File Operations.

Figure 5.23(c) presents evidence that KVM 83 guests outperform XEN 3.4.3 guests and that the XEN 3.4.3 privileged guest Domain0 performs worse than the native host when operating on file metadata. This indicates that the cause of the write bottleneck of KVM 83 with storage cache is due to the inefficient manner in which data is transferred

from the host physical storage to the guest and not due to the handling of file system metadata. Finally comparing Figure 5.23(c) and Figure 5.24(c) another performance regression is illustrated from XEN 3.4.3 to 4.0.1.

5.5 Summary

The aspect of performance within QoS has an important role to play in the provisioning of resources in Cloud Computing. This chapter of the thesis presented the findings of a performance evaluation into the management and utilization of data at IaaS level. The implementation of technologies used in virtual infrastructure has been highlighted as influencing the outcome of resource performance and consequently the QoS provisioned to end users, the competitiveness of a provider in the Cloud ecosystem and likely return on investment of services made available. This subsequently makes the selection of technology a decisive decision for any Cloud provider. The outcome of this work provides quantitative evidence that can enhance an IaaS provider's decision making process and aid in the prevention of SLAs breaches.

The majority of the results of our experiments illustrate that OpenNebula and KVM, relative new comers to the paradigm of distributed systems, perform to a lower standard than Nimbus and XEN. A general theme has reoccurred throughout our performance analysis: the maturity of a particular technology can heavily influences performance. Therefore it can be concluded that the findings of this work advocate mature software solutions due to direct correlation with performance. This should be put into context with the contemporary feature sets these new technologies provide, which one could argue, are more appropriate or specific to the usage scenarios of a Cloud environment. As a consequence this trade off between performance and feature set should be factored in when making any decision on whether to use the technology evaluated here in.

Table 5.3. Cloud Infrastructure Performance Results Summary

Technology	**Benchmark Wins**
OpenNebula	0
Nimbus	2
KVM 83	3
XEN 3.4.3	19

The implications of our results draw attention to the impact performance overheads

have on the adoption of Cloud technology. OpenNebula for example, has a comparatively limited aptitude to react to changes in demand in a stochastic and highly dynamic environment and accordingly Nimbus would be more appropriate in this scenario. Data management services such as Amazon's S3, when considered not economically viable or where the cost of these Cloud services are not completely transparent and vary significantly [151], could result in data having to be stored and accessed within a local VM image. The incapacity of KVM guests at writing and reading data could be a limiting factor for applications that access large quantities of data locally and conversely XEN would be a superior choice here. Additionally, the next Chapter discusses the contextualization of Cloud applications which is reliant on the performance Cloud Computing technology to operate in a timely fashion. Thus the findings of this Chapter can be used to support the contextualization process.

Chapter 6

Contextualization

6.1 Introduction

One of the key questions surrounding Cloud Computing adoption is whether there is real benefit to migrating to the paradigm. In the previous Chapter the issues surrounding performance were discussed in light of enabling more applications to be deployable on Cloud infrastructures. In this chapter issues surrounding scalability are discussed with regards to how best to configure and develop Cloud applications for this end. In addition, the question of how easy are Clouds to set-up and use is addressed. Outages and lack of implemented fault tolerance mechanisms across geographic locations provides motivation that Clouds are no easier to configure than standard Distributed Systems[1].

This chapter discusses the automation of Clouds and currently available tools to configure them. The concept of contextualization and re-contextualization are introduced to address the short comings of these tools for general purpose applications across the entire Cloud service software stack. These concepts are developed into mechanisms and implemented as part of collection of tools to configure and reconfigure VM base images, for the purpose of aiding in the deployment of Cloud applications from the point of view of the developer. The performance of these implementations are evaluated to assess usage and applicability in real world multi-user environments.

The remainder of this chapter is organised as follows: Section 6.2 discusses issues

[1]Latest outage raises more questions about the Amazon Cloud: `http://gigaom.com/cloud/latest-outage-raises-more-questions-about-amazon-cloud/`

surrounding the configuration of the Cloud technology. Section 6.3 introduces the concept of Contextualization presenting requirements for and an architecture of a Contextualization Tool. Section 6.4 extends the work on contextualization proposing a runtime Recontextualization mechanism. Finally, Section 6.5 evaluates the performance of the Contextualization Tool and prototype Recontextualization mechanism.

6.2 Configuring The Cloud

This section of the chapter discuss the heritage of Autonomic computing in Clouds, some tools used to reduce the complexities of configuring Cloud applications and configuration issues limiting the use of Cloud platforms.

6.2.1 Autonomous Cloud Computing

As many computing paradigms share similar concepts and underlying principles, it is easy to confuse Cloud Computing. Cloud Computing integrates the ideas from several paradigms making its application powerful in a vast number of scenarios. Many of the principles that underlay Cloud Computing can be linked to Autonomic computing but are applied in different scopes.

Autonomic computing has been a topic of interest for many years with research starting in 2001 by IBM's autonomic computing initiative. The initiative aimed to create a self-managing computing system, capable of handling increasingly complex systems. Autonomic computing has been defined as:

Definition *Autonomic Computing* "Computing systems that can manage themselves given high-level objectives from administrators." [144]

Autonomic computing involves the creation of systems that run diagnostics checks and compensate for any irregularities that are discovered. Multiple closed control loops adjust the system to maintain its state within a number of specific bounded criteria. The ever growing complexity of distributed systems provide motivation for the use of autonomous systems as manual control is expensive, prone to errors and time consuming. Autonomic computing reduces the need for system maintenance with aspects such as security or software configuration maintained in an unattended fashion. Administrators instead of controlling entire systems by hand, define general rule based policies that guide "*self-**" management processes in four functional areas:

- **Self-Configuration**: Automatic configuration of components

- **Self-Healing**: Automatic discovery, and correction of faults
- **Self-Optimization**: Automatic monitoring and control of resources to ensure the optimal function with respect to the defined requirements
- **Self-Protection**: Proactive identification and protection from arbitrary attacks

Cloud Computing incorporates many features and elements of autonomic computing. Cloud providers use self-regulating system to reduce the complexity of configuring virtual resources and the Cloud service stack, enable transparent fault tolerance through resource monitoring, optimisation of resource usage through the consolidation and migration of VMs and pro-actively defend against attacks with Intrusion Detections Systems (IDS). Without such measures in place, providers would not be able to pass on the cost benefits to its customers through the economy of scale they provide. A distinguishing feature of Cloud Computing over Autonomic Computing is that it requires in most cases that the End-User have an understanding of the infrastructure and software that supports it, while the virtual resource interface presented in Clouds does not. Due to the size of Cloud environments Self-Configuration plays a critical role in enabling the other self-management processes and the deployment of anything more than the most simple of Cloud applications. As a result, the remainder of this subsection concentrates on the use of Configuration Management Tools and the issues surrounding the configuration of application software stacks in Clouds.

6.2.2 Configuration Management Tools

Configuration Management (CM) is used in many disciplines from civil engineering to military applications such as weapons system development and has many definitions. In this thesis within the context of distributed systems, the following definition is used:

Definition *Configuration Mangement* A management process that focuses on establishing and maintaining consistent system performance and functional attributes using requirements, design, and operational information throughout a system's life-cycle. [3]

Configuration Management is a large area of work. For the research presented in this Chapter, particular attention is placed on a set of tools that provide similar automation functionality to that presented here in on the topic of Contextualization but that has not been designed in the context of Cloud Computing and thus tailored for use in the dynamic environments that Cloud applications operate within. Additionally, these tools have issues of convergence where the continuous reconfiguration of software does not lead to any

stable or desired state. These same issues are applicable to the research discussed later in this Chapter on the topic of Recontextualization and has an impact on the decision making processes within a Cloud environment. For example, if a decisions is made to migrate a Cloud resource and reconfigure it for a new environment, only for the problem for which it was migrated not to be resolved, a perpetual state of Recontextualization could occur.

There are three generalised approaches to configuring the Cloud service software stack:

- **Manually**: Configuring each software component by hand. Not feasible in large scale Cloud deployments, is time consuming and error-prone.
- **Pre-Configured Static Images**: A common approach where a set of images are set-up once and reused. A brittle approach where changes to the configuration of the images have to be made to all instantiated VMs with no way of propagating the changes from previous image versions currently in use.
- **Configuration Management Tools**: Tools or software approaches that apply CM to automate the configuring of an image or instantiated VM. This provides enhanced control and flexibility.

Configuration Management Tools provide a number of benefits including i) the reproducibility and industrialization of software configuration, ii) the continuous vigilance over running systems with automated repairs and alert mechanisms, iii) enhanced control over and rationalisation of large scale deployments and iv) the ability to build up a knowledge base to document and trace the history of a system as it evolves.

This subsection of the thesis discusses the concept of Configuration Management for the automated deployment of Cloud applications software dependencies at the PaaS level. The feature sets and comparison of three configuration management solutions: CFEngine 3 [40], Puppet [199] and Chef [41] are presented. Both systems can be used to automate infrastructure deployment covering thousands of machines.

6.2.2.1 CFEngine

The CFEngine [40] project provides automated configuration management of large networked systems. CFEngine can be deployed to manage many different types of computer system such as servers, desktops and mobile/embedded devices. The project was started in 1993 by a post-doc, Mark Burgess at Oslo University, as a way to automate the management of dissimilar Unix workstations[2] via the abstraction of platform differences using

[2]Details of the initial version of CFEngine can be found in an internal report at: `http://www.iu.hio.no/~mark/papers/cfengine_history.pdf`

a domain specific language. In [34] the foundations of self-healing systems were developed by Mark Burgess and as a precursor, heavily influenced the ideas of Autonomic Computing developed later by IBM.

CFEngine uses decentralised, autonomous software agents to monitor, repair and update individual machines. It is comprised of a number of components with varying responsibilities:

- `cf-promises`: Used to pre-check a set of configuration promises before attempting to execute them.
- `cf-agent`: An agent that manipulates system resources to enact change.
- `cf-serverd`: A server able to share files and receive requests to execute existing policy on an individual machine.
- `cf-execd`: A scheduling daemon, executing and collecting the output data from cf-agents.
- `cf-runagent`: A helper program that communicates with `cf-serverd` for the purpose of requesting the update of `cf-agent(s)` existing policy and is used to push out changes to CFEngine hosts.
- `cf-report`: Generates summary and other reports in a variety of formats.
- `cf-know`: An agent used for rendering documentation as a 'semantic web' from system knowledge.

The components operate over four phases of system management, which are based on transactional changes:

- **Build**: A template of proposed promises is built for the machines in an organization. If the machines keep these promises, the system function as anticipated.
- **Deploy**: Implement previously decided policy via pushing out changes to agents.
- **Manage**: Autonomous agents manage unplanned system events. Rare events that cannot be dealt with automatically set off alarms for human intervention.
- **Audit**: System report generation for determining what changes were made by agents and if they were policy compliant.

At the core of CFEngine lies the idea of convergence [35], where the final desired state of the system is described instead of the steps needed to get there. This enables CFEgine to run whatever the initial state of the system is with predictable end results. The downside to this approach is that only statistical compliance or best effort can be achieved with a given configuration policy, where by a system can not be guaranteed to end up at a desired state but slowly converges at a rate defined by the ratio of environmental change to the rate at which CFEngine executes.

Puppet [199] and Chef [41] are two other configuration management systems widely used as replacements for CFEngine and have tried to combat particular short comings in CFEngine's design and ideology when used in some specific use-cases.

6.2.2.2 Puppet

Puppet was forked from CFEngine and provides graph-based and model driven approaches to configuration management, through a simplified declarative domain specific language that was designed to be human readable. The model driven solution enables the configuration of software components as a class, a collection of related resources where a resource is a unit of configuration. Resources can be compiled into a catalogue that defines resource dependencies using a directed acyclic graph. A catalogue can then be applied to a given system to configure it.

6.2.2.3 Chef

Chef, a fork of Puppet, rose out of the Ruby-on-Rails community out of dissatisfaction with Puppet's non-deterministic graph-based ordering of resources. This is useful for bringing a system from its current state into compliance with a specified state and where ongoing configuration changes are more important than the initial provisioning. In contrast, Chef places emphasis on starting up services from newly provisioned cleans systems, where the sequence and execution of configuration tasks is fixed and known by the user. This makes Chef particularly well suited to the paradigm of Cloud Computing where VM instances are short lived and new instances are spawned from a newly provisioned base image. Chef uses the analogy of cooking and creates "recipes" that are bundles of installation steps or scripts to be executed.

6.2.2.4 A Comparison

All the tools share a common ancestry with CFEngine 3 and have been designed specifically with configuration management in mind. The tools are open source, provide text-

based interfaces and are based around the client-server model. The tools differ in a number of ways: support for Windows in full is only provided in CFEngine 3 which is also written in C and follows the GPL license. Puppet and Chef have limited partial support for Windows with regards to what aspects of the Operating System can be configured and both are written in Ruby. Finally, Chef does not provide a declarative domain specific language and instead uses extensions of the Ruby programming language to describe configuration. This can be of benefit but there is a trade off between clarity in defining the configuration and power to configure as needed.

Table 6.1. Comparison of Configuration Management Tools

Tool	**Language**	**Platform Support**	**Domain Specific Language**
CFEngine	ANSI C	Windows, Mac OS X, BSD, Linux	Yes (Declarative)
Puppet	Ruby	Windows (partial), Mac OS X, BSD, Linux	Yes (Declarative and Imperative)
Chef	Ruby	Windows (partial), Mac OS X, BSD, Linux	No (Imperative)

The performance and scalability of Puppet and Chef are limited due to the language they are written in and the maturity of the solutions. There are still several issues with the scalability of Puppet out of the box[3]. Chef scalability still remains to be fully explored[4] and several issue are not solvable without a familiarity with generalised scaling techniques[5]. On the other hand, CFEngine is written in `C` and its decentralised architecture has recorded deployments by enterprise users of over 10,000 host nodes.

Although Chef is well suited to Cloud Computing with its emphasis on configuring systems from scratch, it does not resolve all the issues surrounding configuring an application in a dynamic environments and has been particularly tailored to the deployment of Ruby on Rails applications in Clouds. The next subsection provides further details on these issues in the context of Cloud Computing and introduces the idea of contextualization as a mechanism to provide VMs with identities for the purpose of configuring clusters or resources for general purpose use.

[3]Puppet scalability:`http://projects.puppetlabs.com/projects/1/wiki/Puppet_Scalability`
[4]Chef scalability: `http://lists.opscode.com/sympa/arc/chef/2011-08/msg00000.html`
[5]Concurrency in Chef `http://lists.opscode.com/sympa/arc/chef/2012-01/msg00422.html`

6.3 Contextualization Tools

Modern Virtualization technologies enable rapid provisioning of Virtual Machines (VMs) and thus allow Cloud services to scale up and down on-demand. This elasticity [94, 152] comes with a new set of challenges for dynamic service configuration. Contextualization is a set of processes and mechanisms that enable a service to scale elastically alongside the resources and software that support it through the orchestration of these dependencies toward the common goals of the service. The focus of the research is on horizontal elasticity where scaling is achieved by adding or removing VMs to a service during its operation. The related case of vertical elasticity, i.e. application scaling through VM resizing, is much easier from a contextualization perspective. For horizontal elasticity scenarios, predefined yet flexible contextualization mechanisms enable the VMs of a service to self-discover and communicate. For clarity, the definition of contextualization is as follows:

Definition *Contextualization* is the autonomous configuration of individual components of an application and supporting software stack during deployment to a specific environment.

Traditionally contextualization involves the manipulation of VM images during the development of a Software as a Service (SaaS) solution and requires the complex and time consuming configuration of software within the images.

Three main challenges are identified on enabling elasticity through contextualization where VMs are added and removed during service operation:

1. Contextualization support offered by the Platform as a Service (PaaS) layer, to replace the traditional approach that requires complex and time consuming manipulation of VM images as part of development of each Software as a Service (SaaS) solution.

2. Contextualization of services that are deployed across multiple Infrastructure as a Service (IaaS) providers.

3. Contextualization with support for functional requirements such as secure network overlays and the incorporation of licence-protected software in services.

The implementation of Contextualization Tools are highlighted as part of the OPTIMIS Toolkit [74], a set of software components for simplified management of Cloud services and infrastructures but are generic enough to be used in other IaaS toolkits. The

feasibility of the contextualization mechanisms is demonstrated in a three-tier web application scenario, where the database and application server tiers both are replicated and scaled elastically. The contextualization mechanisms enable elasticity for this application such that instances of database and application server can be transparently added and removed from the service during operation. Results are also provided that demonstrate the scalability and performance of the mechanism in a multi-user Cloud environment.

6.3.1 Contextualization Challenges

It could be argued that contextualization in Cloud Computing remains a highly pervasive key technological requirement of any Cloud service, where elastic resource management is critical to the on-demand scalability of a service. The holistic nature of the services deployed on Clouds makes it difficult to provide flexible generic and open PaaS tools without limiting heterogeneity of supported services. Three inherent challenges are identified to providing elasticity through contextualization where VMs are added and removed during service operation.

The first challenge to overcome is the complete contextualization of Cloud services across all classifications [254] within the Cloud ecosystem: SaaS, PaaS, and IaaS. This pertains to low-level contextualization of virtual resources, as found in IaaS providers, where virtual devices require context to enable VMs to be bootstrapped to existing virtual infrastructures. This approach has been partially explored by Reservoir [209]. In addition, the contextualization of software dependencies that supports a deployed service in a PaaS provider needs to be scalable. Finally, the service itself must be developed in such a fashion that enables scalability.

The second challenge to overcome is contextualization across multiple IaaS domains for reasons of interoperability. Many IaaS providers, such as Amazon Web Services [5], offer PaaS services that are not interoperable with those of other providers. In these PaaS services, contextualization is performed as part of service development, which makes the process customized to a single provider only. This presents an opportunity for contextualization at service deployment time only, thus enabling interoperability between IaaS providers by not having to rely on these platform services. Contextualization at deployment time only, incurs additional challenges in relation to the re-contextualization of resources during runtime for the accommodating of service elasticity and on-demand scalability, whereby platform services must be self-discovering and autonomous. This adds complexity to the contextualization mechanism that must be used to support the software dependencies of a service at the platform level and is discussed later in this chapter.

The third challenge pertains to a set of functional requirements for real world Clouds and their impact on contextualization. Notable among these are end-to-end security through contextualization mechanisms that support a Virtual Private Network (VPN) overlay and software license management systems. Both of these have unique contextualization requirements: contextualization must be secure with no VPN keys stored unless in use and contextualization that is able to accommodate license protected software with licensing tokens.

6.3.2 Related Work

Approaches to contextualization vary considerably depending on the nature of the application or virtual appliance. In the Grid community, research into the effective use of Cloud Computing for academic use and the implications on contextualization have been explored [142, 171]. The approach to contextualization by the Nimbus Project [177] in [142] is to integrate heavily with the Globus Toolkit [88], limiting the general applicability of the contextualization solution to a small number of use cases. In addition, recontextualization of resources is considered to be a necessity unlike the approach outlined in this research. In [171], details of a mechanism for the contextualization of scientific virtual appliance are discussed for the purpose of replicating Grid services. An approach is proposed that exposes users to a high level declarative language in XML for performing typical steps in the deployment of a scientific application. The goal of this contextualization mechanism differs from our own in that we seek to enable elasticity rather than fault tolerance of Cloud resources and again the proposed solution is specific to a single use case from the academic community.

Finally in [73], the challenges and techniques of contextualization are presented for a cluster environment at the Clemson University. The paper describes low level issues of contextualization and makes some recommendations on contextualization from practical experience. Image-level contextualization and the problems associated with its automation and complexity are discussed in detail but no solution is provided. The research of this thesis is differentiate from the above through the application of contextualization in several Cloud scenarios from the OPTIMIS project by providing viable solution to these problems.

6.3.3 Requirements and Architecture

6.3.3.1 The OPTIMIS Toolkit

The contextualization research is performed in the context of the OPTIMIS Toolkit [74], a set of software components aimed to simplify and optimize the construction, deployment, and operation of services (at the SaaS and PaaS level) as well as the operation of virtualised hardware needed to deliver these services (IaaS level). Although OPTIMIS targets scenarios and capabilities somewhat different to the ones provided by current PaaS and IaaS providers, the OPTIMIS contextualization requirements are general and the solution should be applicable in a wide range of Cloud providers. The research on contextualization is presented in light of OPTIMIS Toolkit and project to facilitate the readers understanding but it should be clarified that it is purely the work of the thesis author.

The two beneficiaries of the OPTIMIS Toolkit are Infrastructure Providers (IP) who operate the infrastructure resources (IaaS level) required to provision services and Service Providers (SP) who deliver these services to end users by combining the PaaS and SaaS roles. An OPTIMIS service is functionally accessible by end users over a network. A service is also virtualised such that it is provisioned by means of VMs and elastic in the sense that the number of VMs it uses can vary over time. In overview, the service lifecycle has three steps:

1. Construction where the SP constructs the service (by implementation, orchestration of existing services, and/or use of licence-protected legacy software) and packages it as a set of VM images described in a service manifest.

2. Deployment where the SP initiates the provisioning of the service in an IP.

3. Operation where the IP manages the VMs and related infrastructure resources whereas the SP monitors and controls the application level aspects of the service.

The OPTIMIS Toolkit is a set of stand-alone components that can be adopted and configured to support a range of Cloud deployment models. The targeted scenarios include: private Clouds where SP and IP are combined, Cloud bursting, where private Clouds utilize external IPs; Cloud federation, i.e., collaboration between IPs; Multi-Cloud deployment, where an SP interact with more than one IP; and brokering, a variant of the latter where a third-party broker mediates between SPs and IPs.

The simplest scenario is a *Private Cloud*, where the SP and IP are within the same organization and cooperate to provision resources for services using an internal infrastructure. In *Cloud bursting* scenarios, the IP of a private Cloud manages peaks in demand for

a service by deploying additional VMs in another IP. Another possible deployment model is a *Federated Cloud*, where an IP provides resources for an SP on behalf and across a collective of IPs working in collaboration, while being the single point of interaction for the SP. In contrast, in a *Multi-Cloud* deployment scenario, an SP directly negotiates with and deploys part of a service across multiple IPs but no collaboration occurs between these. An additional scenario is *Cloud Brokerage*, where a third-party broker acts as an intermediary between SPs and IPs, potentially also providing added value through its knowledge base.

In addition to support for multiple deployment scenarios, one notable feature, common for all toolkit components, is self-management capabilities (for services, VMs, data, SLA protection etc.). These self-management decisions are made not only with performance-related criteria but also include non-functional aspects. All resource management actions are taken by balancing performance against *TREC*, namely trust (in the IPs and SPs based on previous experiences), risk (of resource and service failure, SLA violations, etc.), ecological aspects such as energy-efficiency and compliance to green legislations, as well as the cost of provisioning services and VMs. To manage the complexity of this multi-objective service provisioning challenge, the OPTIMIS Toolkit has a modular and layered architecture with a *Basic Toolkit* that provides monitoring and assessment engines for the TREC factors and security capabilities. On top of that, a set of *SP Tools* handles e.g., deployment of services, data transfers, and software licenses, whereas *IP Tools* provides basic functionality for infrastructure management, e.g., admission control, management of VMs and data, fault tolerance, and autonomic elasticity control. A in-depth description of the OPTIMIS Toolkit and the motivations for it is beyond the scope of this research and can be found in [74]. Figure 6.1 shows the tools that make up the OPTIMIS toolkit.

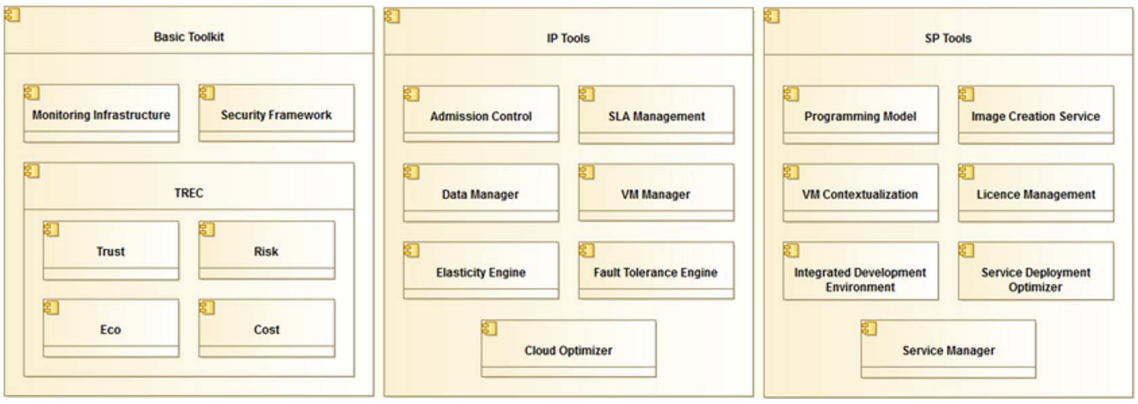


Figure 6.1. Componets of the OPTIMIS Toolkit architecture.

With particular interest from a contextualization perspective are the tools the Service

Deployment Optimizer (SDO) in the SP and the Cloud Optimizer (CO) in the IP. The role of the SDO is to coordinate the service deployment process. Based on the service description in a manifest, including VM images and deployment constraints, the SDO identifies suitable IPs, assesses the Trust, Risk, Eco-efficiency and Cost (TREC) factors related to the service, filters out unsuitable deployment targets, and negotiates terms of use with suitable ones. After selecting the best IP(s), the SDO contextualizes the VM images prior to deployment. Finally, the contextualized VM images are uploaded to the selected IP and the service is started.

The CO has a similar coordination role in the IP as it handles all VM launching requests. New VMs can be started upon deployment of a new service, restarted as part of fault recovery, or booted due to elasticity, i.e., service scale up caused by increasing workload. For each VM launch request, the CO initially decides whether the VM should run in the local infrastructure of the IP or outsource to an external provider (bursting or federation). In the latter case, the SDO functionality for selection of partnering IPs and deployment negotiation is reused. Notably, as VM images boot, they can, thanks to the preparatory steps performed by the SDO, contextualize themselves without synchronizing with any component external to the service, in the SP or elsewhere.

The OPTIMIS service lifecycle management and support for many Cloud deployment models, highlight the need to address the three main contextualization challenges previously discussed. The first, contextualization support as part of a PaaS tool used during service deployment (as opposed to SaaS solutions for service development). The second, contextualization that supports service deployment across multiple IaaS providers and finally support in contextualization for functional aspects such as secure cross-domain networking and use of license protected software. The following is a list of OPTIMIS specific contextualization requirements:

- The VMs of a service need to be able to communicate with each other seamlessly. The different Cloud deployment models supported complicate this, as VMs may be spread across multiple IPs, i.e., several network domains.

- A service must be able to make use of license protected software packaged inside VMs. There is hence a need for a mechanism to propagate software license credentials.

- Services are elastic, it must be possible to launch multiple running VM instances from a single VM image.

- Elasticity also adds the additional complication that not all VMs are launched at the

same time. The exact number of VMs used by a service varies dynamically during operation and is not know in advance.

- Similarly, due to the different Cloud deployment models supported, it is not known a priori in which network, or even which IP, the VMs of a service are hosted.
- The incorporation of license tokens in VMs and need for cross-domain service networking raise several security concerns. To summarize these, contextualization mechanisms must be designed in a manner that does not expose assets such as services, the networks connecting them, and software licenses to exploitation.
- Security information (keys etc.) needed for the establishment of a VPN overlay should not be stored in the VM image nor found on uninstantiated VMs.

There are several problems to solve regarding contextualization of OPTIMIS system level components, such as those associated with license management and Cloud security. The following subsections outline these issues in more detail:

6.3.3.2 License Management

Access to licenses for authorizing the execution of an application in a Cloud beyond the administrative domain of the site running the license server usually leads to applications aborting during start-up because of unreachable license servers, e.g. due to firewall issues. In OPTIMIS a prototype for software licensing is used and was developed in the European project SmartLM [37], which provides licensing technology for location independent application execution. Separation of authorization for license usage and authorization for application execution on the one hand and software tokens that carry the authorization information on the other hand provide the necessary flexibility for licenses following applications into Clouds. It is one of the tasks of the VM Contextualizer to retrieve and embed a license token into the VM hosting the application. In case of multiple applications in a VM, required e.g. for a workflow, the VM Contextualizer assures that all required licenses are in place when the applications start up. No communication between the application and the license server that issued the token is required at runtime.

Additional approaches are planned to be implemented enhancing the SmartLM solution: (i) dynamic deployment of a trusted instance managing a number of tokens for one or multiple applications and (ii) dynamic deployment of a full license service with a subset of the licenses available at the home organisation of the user. The configuration of the dynamically deployed license service will be managed by the VM Contextualizer.

This approach is especially useful when the same Cloud resources are used over a longer period of time for running license protected applications.

6.3.3.3 Cloud Security

Each instance of a VM requires specific security customisations based on the service it provides and its threat profile. For example, the firewall rules specific to a web server VM is different from that of a database VM and these variations are handled by the contextualization tools. In addition, the OPTIMIS Data Manager provides a means of provisioning secure encrypted storage devices for VMs, where the decryption keys are stored outside an IP. The specifics of these secure device configurations are different across various VMs and are set by the contextualization steps. Various Identity and Access Management (IdAM) components that need to be installed, along with policies specified at the VM endpoints, are also set by this component. If required by the end user or SP, other security software like intrusion detection and prevention software and VPNs can also be instantiated and customised by similar mechanisms. Figure 6.2 illustrates some possible VPN scenarios for securing inter-Cloud communication.

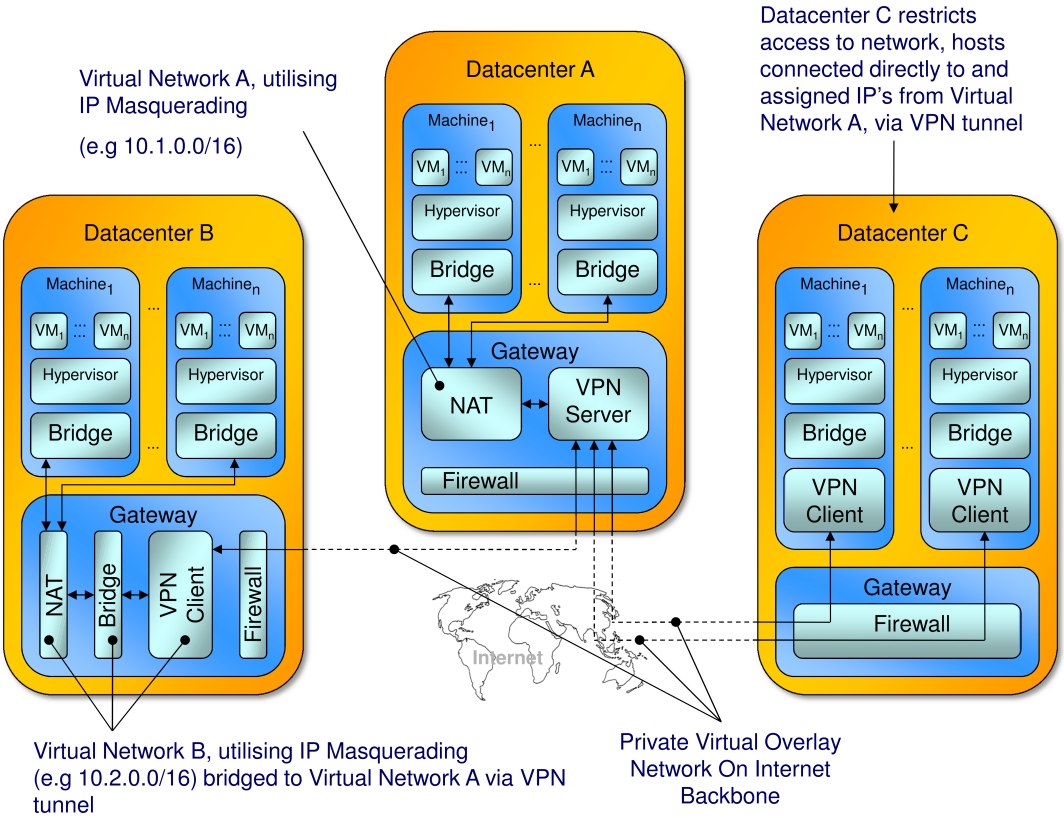


Figure 6.2. Virtual Private Networks - Securing Inter-Cloud Communication.

6.3.3.4 Architecture

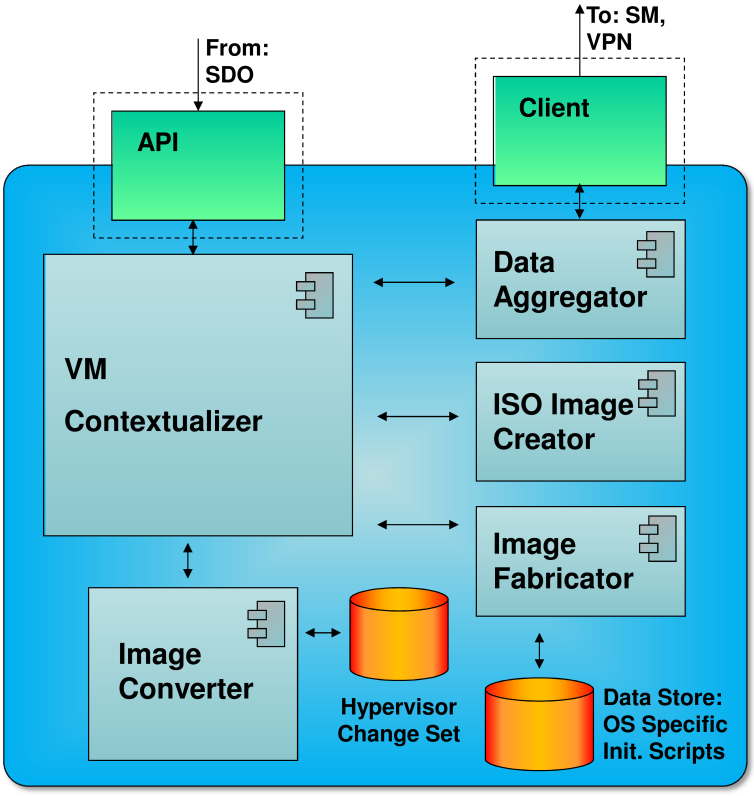


Figure 6.3. Deployment time contextualization architecture.

The contextualization tools are comprised of a core component, the VM Contextualizer and a repository of scripts. This component provides an interface for consuming service context data, such as: security certificates, VPN hostnames, VPN DNS and Gateway IP addresses, mount points for network data stores, monitoring manager hostnames, offline software license tokens and a list of software dependencies. The VM Contextualizer provides two capabilities. The first is a bootstrap mechanism to prepare a VM image for receipt of context agnostic of the operating system type used. The second capability provides a mechanism for creating ISO CD ROM images that contain context data and a data processing script for the manipulation of the data into a format suitable for consumption at runtime when the ISO image is mounted. The VM Contextualizer mounts a VM image and modifies it to include an assortment of bespoke scripts that interact with the guest operating system, service and service dependencies at boot time, preparing the image to receive context in a reusable fashion. At run time these scripts access contextualization data held within this ISO image, as per the OVF recommendation [192], giving a VM its identity. Due to the OVF recommendation, this same storage medium is used by a number

of projects such as Reservoir [207] and Contrail [52] but neither of which focus on the platform level as is the case with the research presented in this Chapter.

The inclusion of the ISO image as a mechanism to store contextualization information provides a facility to separate the contextualization data from the VM image. This removes the time consuming need to create multiple unique VM images for each VM that is required to be contextualized, while also improving the security of the contextualization process as security certificates are not stored in the VM image itself but instead stored in the ISO image when it is dynamically generated. The inclusion of a script to process the contextualization data provides an approach to store the data agnostic of the operating system, service and service dependencies used.

Figure 6.4 illustrates the instance-level contextualization process of a VM at the beginning of its execution. During the boot sequence of a VM the contextualization tools mount an ISO CD image that contains contextualization data and a script to process this data into a usable form. This script communicates through a known interface to the OS specific contextualisation scripts embedded in the VM image. These OS specific scripts manipulate configuration files of associated OPTIMIS components and software dependencies that support the end user's service(s), setting their context. The scripts can remain in a daemon-like mode for a component or software dependency that requires continuous updates to its context, enabling limited dynamic reconfiguration and instant discovery of elastic resources.

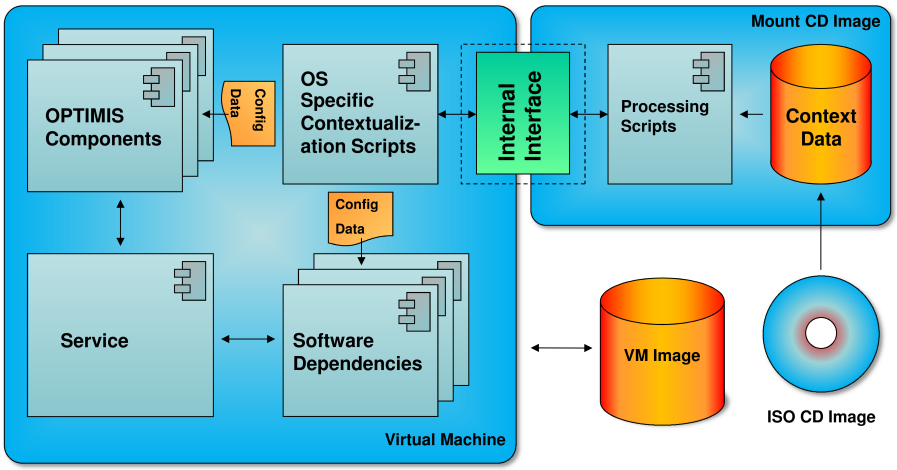


Figure 6.4. Interaction between VM image and ISO Image at run time.

The VM Contextualizer uses parts of QEMU [200], a generic and open source machine emulator and virtualizer to manipulate images. QEMU provides a tool named "qemu-img" that enables the conversion of virtual machine images. In addition, Linux system tools (such as "`mount`", "`iosetup`" and "`kpartx`") are used to mount VM

images as loop devices for write manipulation. For the creation of ISO images, the Linux system tool "`mkisofs`" is used in addition to the previously mentioned tools to create and modify ISO images.

6.3.3.5 Detailed Design

The VM Contextualizer is written in Java and compiles into a Jar file. In the context of OPTIMIS the SDO makes use of this file in its class path. The Jar provides an API to the functionality of the VM Contextualizer and has been designed to be multi-threaded and asynchronous, enabling concurrent services to be contextualized at any one time. Table 6.2 outlines the methods available in the API.

Table 6.2. VM Contextualizer API

Operation	**Input**	**Output**	**Description**
contextualize-Service()	Service-Manifest	Image-Identifier	Given an XML representation of the ServiceManifest containing the URI(s) of VM image(s), this function prepares and/or converts an VM image stored on the local files system into a contextualization ready version in addition to creating ISO CD images on a per VM instance basis. On completion, the function returns a ImageIdentifier object containing a list of new image URIs.
contextualize-ServiceCallback()	ServiceId	Progress	Given a ServiceId, this function returns a Progress object containing the current phase and the phases % completion of a contextualizeService operation.

In OPTIMIS the Service Manifest plays a critical role in contextualization of a service and defines what contextualization processes are required to tailor a server to a specific IP. The Service Manifest is an XML document used to describe and define a service for deployment and is used throughout the OPTIMIS lifecycle. The VM Contextualizer takes the Service Manifest as input when invoked, parsing it for the VM images that it is required to operate on. The Service Manifest provides both the OVF hardware definition of the VMs and a specific SP side extension that contains the contextualization requirements.

Figure 6.5 shows the dependencies between the sub-component packages that make up the VM Contextualizer as illustrated in Figure 6.3. At the heart of the VM Contextualizer is the `Core` package that coordinates the other subcomponents and contains the core logic of the Contextualization Tool.

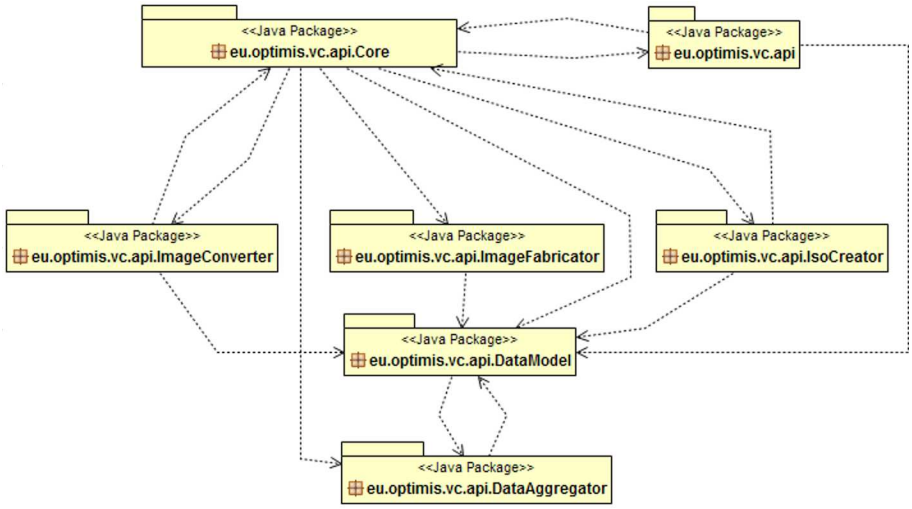


Figure 6.5. Overview of Package Dependencies in the Contextualization Tools.

The `Core` package coordinates with the `ImageConverter`, `ImageFabricator`, `IsoCreator` and `DataAggregator` packages and presents its functionality via the `Api` meta-package as outlined in Table 6.2. All packages depend on the `DataModel` package that contains a collection of objects with inheritable attributes. These objects form a structured abstraction of the data used during contextualization for the purpose of enabling data exchange between sub-components. A class diagram of the VM Contextualizer is presented in Appendix A (Figure A.1) that provides details on the data model used.

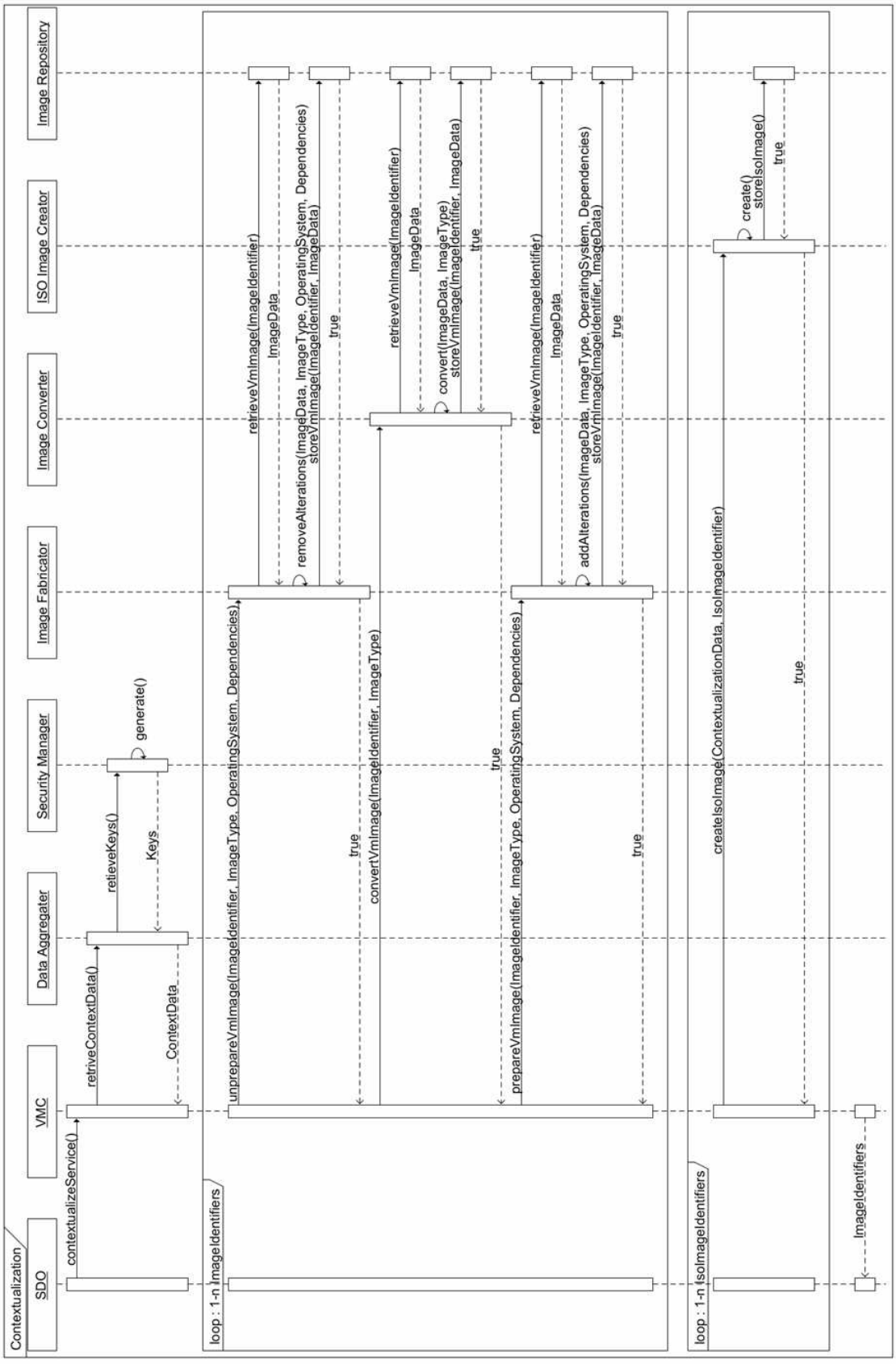


Figure 6.6. Sequence Diagram of deployment time contextualization.

Figure 6.6 describes the interactions of the VM Contextualizer, coordinated by the `Core` package, with its sub-components and other components in the OPTIMIS Toolkit at the SP level. From the sequence diagram, it can be seen the SDO initializes the contextualization of a service by supplying the Service Manifest to the VMC. The VMC then calls the Data Aggregator to parse the service manifest and call the Security Manager to generate keys. The contextualization data is then used by the VMC to call the Image Fabricator to prepare a VM image for receipt of this data at runtime.

When conversion of a VM image is required the Image Converter is called and if the image has been prepared previously by the VMC, the Image Fabricator is invoked twice, before and after conversion to un-prepare and then prepare again the image for receipt of contextualization data. Calls to the Image Fabricator and the Image Converter access the local file system based Image Repository. After all VM images have been prepared and/or converted the VMC invokes the ISO Image Creator to create ISO CD image files containing contextualization data on a per instance basis. The resulting files are then stored in the Image Repository. Finally the VMC returns a list of Image Identifiers represented as URIs (both VM and ISO) to the SDO that correspond to the contextualised service.

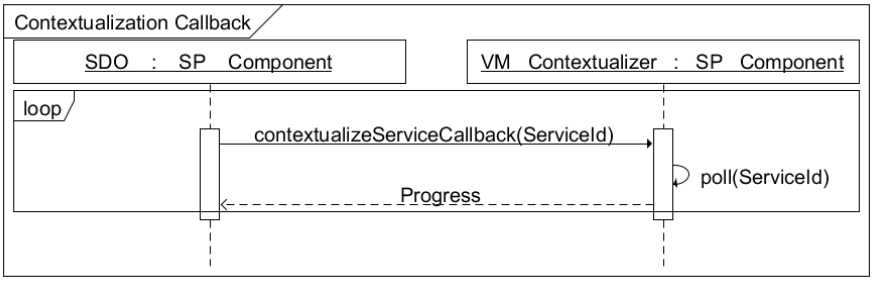


Figure 6.7. Asyncrounous progress callback for deployment time contextualization.

To enable the progress of the VMC to be monitored by SDO in the case of OPTIMIS and presented to an End-User via a GUI, the VMC provides a single callback, as illustrated in Figure 6.7, which returns the phase of contextualization and the percentage completion of the phase. The phases returned in order are:

- Parsing Manifest
- Gathering Contextualization Data
- Decomposing VM Image(s)*
- Converting VM Image(s)*

- Fabricating VM Image(s)*
- Generating ISO Images

Phases marked with an '*' are considered optional and can be opted into via specifying the contextualization requirement in the Service Manifest. The remainder of this subsection describes in detail the roles each sub-component plays during service contextualization.

Sub-component: ISO Image Creator

The ISO Image Creator sub-component manipulates a set of template ISO images. In addition, this sub-component is responsible for the creation of ISO images and the embedding of context data from the Data Aggregator. The ISO Image Creator has access to a single set of context data processing scripts that are agnostic of a VM's operating system, embedded in the ISO image and responsible for reading context data from the ISO image at run time. An internal interface provides access to contextualization scripts embedded in an OPTIMIS enabled VM image.

Sub-component: Image Fabricator

The Image Fabricator manipulates VM images, passed to it by the VM Contextualizer, using Linux system tools. The sub-component installs OPTIMIS VM level components. In addition, software dependencies of a service can be embedded into the image, if the VM Contextualizer supports them. If this is not the case it is assumed that the Service Developer had already setup the context of the services software dependencies within the VM image provided to the VM Contextualizer.

A set of prefabricated operating system specific contextualization scripts, from a data store, are embedded for each operating system with foreseen use at the IP level. The contextualization initialization scripts provide two main functions. The first is to set the context of OPTIMIS components that facilitate running a service in addition to the configuration of the Operating System. This involves setting the context of:

- The OPTIMIS Data Management tools at the VM level and any associated software dependencies, which assist in mounting block stores in a VM at runtime
- The OPTIMIS VPN management component, to enable the VM to communicate with other VMs on a VPN.

- The Monitoring Probes, enabling monitoring data of non-functional service KPIs present in a SLA as requirements to be sent to the SP.
- The License Manager, to enable automated registration of license tokens with software dependencies at the point of VM execution (Offline) or during run time (Online).

The VM's virtual Network Interface Card (NIC) attached to the IP's Infrastructure when DHCP is unavailable, is configured with an appropriate Internet Protocol address by decoding the MAC address of the NIC set by the IP. This contextualization process requires no input from the SDO. The MAC address is encoded so that the first four hexadecimal bytes define the default gateway address once converted to decimal format. The last two bytes of hexadecimal digits define the last two segments of the VM's address on a Class A or B network. Combining the two, the VM address can be obtained by the contextualization scripts during boot time. The following is a worked example:

- MAC address: *0C-A8-01-01-*`02-0A`
- Gateway address: *192.168.1.1*
- VM address: 192.168.`2.10`

The second function of the contextualization scripts is to provide context to the software dependencies of a service. Inclusion of these scripts is optional and is dependent on what software OPTIMIS supports.

Sub-component: Image Converter

The Image Converter transforms images from one format to another for the purpose of supporting interoperability between IPs using dissimilar Hypervisor technology. The Image converter makes use of "`qemu-img`" to convert between image standards. Depending on the Hypervisor and operating system, conversion can require the changes made by the Image Fabricator to be reverted and reapplied. The following Hypervisors are supported:

- Xen
- KVM
- VMware

- VirtualBox

In addition the subsequent image formats are supported:

- Raw
- QCow2
- VDI
- VHD
- VMKD

Besides support for multiple image formats the Image Converter supports copy on write, compression and encrypted images.

Sub-component: Data Aggregator

The Data Aggregator is responsible for the gathering of configuration data from multiple sources. The component parses the Service Manifest (SM) for the following contextualization requirements:

- License Management: License server end-point on a per-service basis and/or SM embedded license tokens on a per VM instance basis.
- Programming Model: SSH keys needed for secure communications and software dependencies on a per VM-type basis.
- Data Management: Data Manager server end-point for mounting VM level block storage on a per-service basis.
- Security Management: Specification of the VPN network topology associated with VM-type.
- Monitoring: Probe end-point for reporting KPIs to that are IP specific attributes sent to the SP during the last phase of SLA negotiation. In addition to parsing the Service Manifest, the Data Aggregator calls the Security Manger to generate keys and certificates on-the-fly.

6.3.3.6 Example Scenario

As is often the case with Cloud Computing applications more than a single VM is needed to provision the resources necessary for a given service load. Configuration management tools can facilitate in the initial configuration phase when deploying the VM but have limited functionality with regards to creating clusters of virtual resources. In this subsection an example web application scenario is examined in light of the OPTIMIS project and contextualization. The scenario provides insight into the complex processes involved in configuring an application that requires cooperative resources.

The application in the scenario is a 3-tier web application comprised of a set of virtual machines including a MySQL database cluster. At the IP level, predefined context from a SP is applied to a VM as it is brought online via contextualization scripts tailored to the modification of MySQL configuration files. This does not require communication with any IP level component and can avoid the complexities of runtime re-contextualization explained later in this chapter. Figure 6.8 illustrates the relationship between the contextualization data and a VM. Red arrows indicate requests while black lines represent the insertion of contextualization and contextualization attributes into a VM. From the figure it can be seen that each layer of the service forms a cluster of cooperating resources that rely on a head node to provide information on the state of available VMs and to balance load.

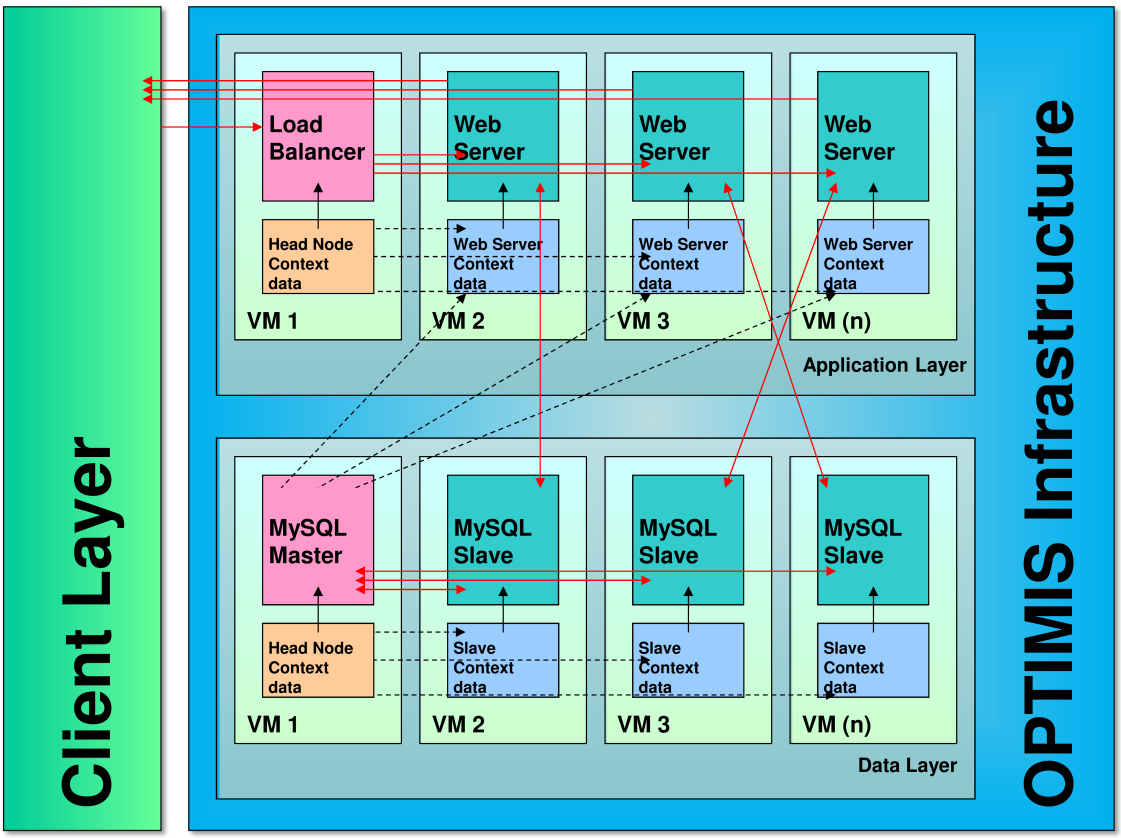


Figure 6.8. Contextualization in a three-tier web application.

Each new VM brought online contains information about the head node to which it is to register for active duty in addition to other VMs to which it must communicate to perform its role. The information is stored within the contextualization data of the VM, which includes a subset of the contextualization data from the head node and can reference other sources of information. These other sources of information, such as an IP registry that contains details of IP specific services such as block storage, can be used to update contextualization data continuously during runtime to enable adaptation to minor environmental changes. This enables VMs to be taken off-line without disturbing the operation of an application.

Taking the Data Layer as an example, the MySQL Master is contextualized to accept Slaves at deployment time via granting replication rights to a special user and by specifying which databases should be replicated in its configuration files. Slave nodes are contextualized to point to the Master hostname and granted access rights via the special user account. As new slaves are brought online the slave registers with the master as shown in Listing 6.1.

Listing 6.1. Slave Registration.

```
mysql> SHOW SLAVE STATUS\G
           Slave_IO_State: Connecting to master
              Master_Host: host.com
              Master_User: slave_user
              Master_Port: 3306
            Connect_Retry: 60
          Master_Log_File: mysql-bin.001804
      Read_Master_Log_Pos: 83207722
           Relay_Log_File: mysqld-relay-bin.001090
            Relay_Log_Pos: 98
    Relay_Master_Log_File: mysql-bin.001804
         Slave_IO_Running: No
        Slave_SQL_Running: No
                      ... : ...
  Seconds_Behind_Master: NULL
```

After registration the Slave synchronises its databases with the masters and is then available for use in the Application Layer as depicted in Listing 6.2.

Listing 6.2. Synchronization Of Slave.

```
mysql> SHOW SLAVE STATUS\G
           Slave_IO_State: Waiting for master to send event
              Master_Host: host.com
              Master_User: slave_user
              Master_Port: 3306
            Connect_Retry: 60
          Master_Log_File: mysql-bin.001804
      Read_Master_Log_Pos: 83207722
           Relay_Log_File: mysqld-relay-bin.001090
            Relay_Log_Pos: 98
    Relay_Master_Log_File: mysql-bin.001804
         Slave_IO_Running: Yes
        Slave_SQL_Running: Yes
                      ... : ...
  Seconds_Behind_Master: 0
```

The MySQL master maintains a list of available slave nodes that can be used to load

balance requests amongst the cluster as shown in Listing 6.3

Listing 6.3. List of available Slaves on the Master Node.

```
mysql> SHOW SLAVE HOSTS;
+-------------+-----------+------+-------------+
| Server_id   | Host      | Port | Master_id   |
+-------------+-----------+------+-------------+
| 1921680101  | slave1    | 3306 | 192168011   |
| 1921680102  | slave2    | 3306 | 192168011   |
+-------------+-----------+------+-------------+
```

The list is fetched by the Web Servers in the application layer during contextualization or on the event that a MySQL Slave is unresponsive enabling a certain degree of reactive re-configuring at the PaaS level within a single IP only. The following section of the thesis discusses how re-configuring of the complete Cloud stack can be achieved across multiple IPs for the purpose of adapting to a changing environment and introduces the notion of Recontextualization.

6.4 Recontextualization

Infrastructure as a Service (IaaS) Clouds are commonly based on virtualised hardware platforms executing and orchestrating self-contained VMs, which are comprised of multiple virtual devices. A Cloud application is typically subdivided into individual components, each component bundled into a specific type of VM. Several VM instances can be started using the same type of VM (using the same master disk image) and each new VM instance is uniquely configured, *contextualized*, with instance specific settings at the early stages of execution. The capacity of the Cloud application can be adjusted by changing the amount of VM instances.

In this work the concept of *recontextualization* is introduced. Recontextualization can be used to adapt to any system changes, including making newly migrated VMs operate properly in the (potentially different) system environment of a new host. This thesis defines Recontextualization as follows:

Definition *Recontextualization* is the autonomous updating of configuration for individual components of an application and supporting software stack during runtime for the purpose of adapting to a new environment.

In this definition, a new environment is considered to be a change in the underlying

physical or virtual hardware, for example when a VM is migrated from one host to another. In addition, changes in infrastructure or platform level services that are in active use by a Cloud application would also be considered to be a new environment. These changes are particularly problematic for live migration of VMs as these migration processes are not self-aware and cannot account for any changes that take place within the a new environment. An analogy to Recontextualization can be made with printers and printer drivers, where by a laptop user moving from one physical location to another (i.e. a new environment) wanting to access different printers at each location, is required to first install a driver for an associated printer if not already present then configure the driver to their needs before the printing device is usable. This process creates a unique identity on the user's laptop for each configured printer and driver, providing a context or setting specific to each environment. The same process is applicable to the virtual devices of a VM and any software dependencies used to support a Cloud application. Without the support of Recontextualization this is a time-consuming manual process and limits the flexibility and scalability of the Cloud. Thus the key benefits of the approach to Recontextualization taken in this thesis are: i) minimal changes to existing Cloud infrastructure, i.e. there is no need to make alterations to the Hypervisor and ii) the preservation of security through the selection of a Recontextualization mechanism that gathers contextualization data from a secured source. In addition to these benefits, there are a number of challenges that must be cover come before Recontextualization can see widespread adoption. These are due in part to a lack of Cloud service provider interoperability and the difficulties in creating an approach to Recontextualization that can be applied to the wide diversity of applications deployed into Cloud Computing environments.

The life-cycle of a Cloud application is comprised of three individual phases as shown in Figure 6.9. The Construction phase refers to the development of a Cloud application making use of platform services and dividing that application into a set of VM images. In the Deployment phase a constructed application is deployed on to suitable infrastructure and finally in the Operation phase the Cloud application is executed. The application can be configured in the Construction phase and contextualized with specifics of a provider's environment in the Deployment phase. Recontextualization offers dynamic reconfiguration in the Operation phase.

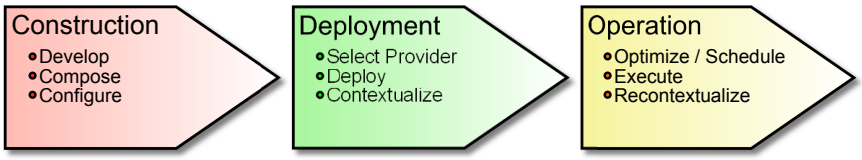


Figure 6.9. The life-cycle of a Cloud application.

Recent work on IaaS systems have a lot in common with the vision of autonomic computing, as outlined by Kephart and Chess [144]. One of the major aspects of autonomic computing that has yet to be realized is self-configuration, the automated configuration and adjustment of systems and components. This work extends state of the art and earlier efforts by introducing runtime recontextualization, enabling adaptation of VM behavior in response to internal changes in the application to which the VM belongs or to external changes affecting the execution environment of the VM. The concept can enable applications at the PaaS level to adapt to different provider application middleware services through the dynamic binding of APIs, enabling the execution of site specific code. This, however, is out of scope in this research. The aims of this research are to present:

1. The concept and definition of recontextualization.
2. The development of an architecture and mechanism for the purpose of recontextualization.
3. A demonstration and evaluation of a recontextualization system.

6.4.1 Requirements and Architecture

6.4.1.1 Problem Statement and Requirements

A motivational factor behind the need for runtime recontextualization stems from VM migration in Clouds [31, 274]. Using migration, a VM can be transferred from one physical host to another without explicitly shutting down and subsequently restarting the VM [46]. The entire state of the VM, including e.g., memory pages, are transferred to the new host and the VM can resume its execution from its state prior to migration. As a consequence of this, no contextualization is triggered again when the VM is resumed, as the level of abstraction provided by Virtualization is insufficient for platform services. In this research migration from and to identical Hypervisor technology is considered, interoperable migration is out of scope but is considered in [159]. As presented in [74], there are several different Cloud scenarios:

- Bursting - The partial or full migration of an application to a third party IaaS provider, this may occur when local resources are near exhaustion.
- Federation - The migration of an applications workload between a group of IaaS providers, e.g., when a single provider's resources are insufficient for maintaining the high availability of an application through redundancy.
- Brokering - The migration of an application's VMs, e.g., for the purpose of maintaining an agreed Quality of Service (QoS) in the case of an end-user utilizing a broker to select a provider given a set of selection criteria.

In all these Cloud scenarios VM migration is a necessity, e.g., for the purpose of consolidating resources and maintaining levels of QoS. These scenarios have been used to guide the defining of requirements for any potential recontextualization mechanism. The following requirements are considered as imperative:

i. A triggering mechanism for recontextualization on VM migration.

ii. A secure process to gather and recreate contextualization data after migration.

iii. A Hypervisor agnostic solution that maintains IaaS provider interoperability.

iv. An approach that is none pervasive and minimizes modifications at the IaaS level.

A case is made for each of these scenarios requiring recontextualization at runtime. In the Bursting scenario, if an IaaS provider is not obligated to divulge third party providers used for outsourcing of computational resources, an application may end up deployed on to a third party's infrastructure that requires use of their local infrastructure services. A dynamic federation of IaaS providers created during negotiation time that alters during the operation phase requires infrastructure services to be discovered dynamically. The same is applicable in the case of a Broker, knowledge of a providers local infrastructure services is not available during deployment until after the Broker has selected a provider.

The lack of knowledge on the attributes of an IaaS provider's local infrastructure service available during deployment time further motivates this research. An example of such a service that exhibits configuration issues after resource migration is application-level monitoring.

In this example the monitoring service endpoint, to which application KPIs are reported, can be configured by contextualization during the deployment phase of an application's life cycle. However, the endpoint may change during the application's lifetime, either as a result of changes in the local system environment or due to migration of the

application to a new host. This example motivates the need for a mechanism to fetch configuration data during application operation and provide new context to application dependencies, thus *recontextualization*. In the following section, recontextualization is illustrated with service-level monitoring [141] as an example scenario.

6.4.1.2 Example Scenario

A typical Cloud application must be continually monitored during runtime, an example of this is shown in Figure 6.10. Monitoring data can be used for several purposes, e.g., for automatic application scaling or to assess the likelihood and prevent the breaching of a SLA. Application level metrics, know as KPIs, are sent from inside the VM to an external monitoring endpoint for processing.

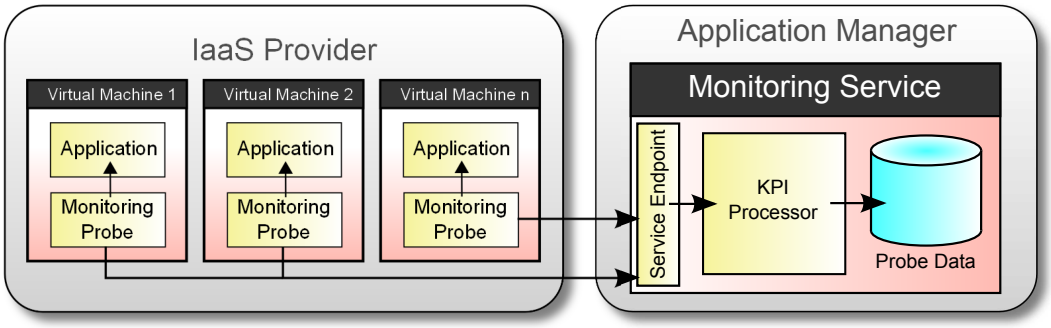


Figure 6.10. Monitoring applications in a IaaS provider.

Each monitoring probe that gathers KPI data must be configured with the endpoint or location of the monitoring service. The endpoint can be associated with a IaaS specific service or a service running at a remote location and depends on what entity within connected Clouds has control over application management. When deploying to a IaaS provider, the endpoint for the monitoring service is configured using contextualization in the Deployment phase, as outlined in Section 6.3. However, in a multi-site scenario the VM maybe migrated to an unknown provider during runtime and must therefore be dynamically recontextualized with a new endpoint in the Operation phase.

6.4.1.3 Recontextualization Approaches

No previous research has considered an approach for recontextualization. Keahey and Freeman [142] present fundamental work on contextualization in virtual clusters and recontextualization is mentioned but deemed out of scope for their work. In this section

several different approaches for contextualization are considered for use in recontextualization. Any recontextualization approach has two major obstacles that must be dealt with; how is recontextualization triggered and where can the necessary context data be found? Below are some approaches for recontextualization, listed and discussed, from the perspective of the above two challenges.

Contextualized direct addressing is based on a known endpoint address that is specified in the initial contextualization phase, as described in Section 6.3. A similar approach is used by Puppet [249]. During operation, this endpoint address is queried for the updated context information. Furthermore, this approach is interoperable and requires no host and Hypervisor modifications, but requires that the end point address is constant when a VM is migrated to other domains. This approach offers no procedure for triggering a new round of recontextualization, and has to rely on periodically polling the endpoint for updates.

Hypervisor network proxying also relies on periodically querying an external endpoint address for context information, but in this method a standard virtual network address is used and the Hypervisor (and associated virtual network management) is responsible for routing this call to a host specific endpoint. This approach, used by Clayman et al. in [47], is transparent to the VM but requires modifications at the Hypervisor level and thus has limited interoperability with commercial Cloud providers.

Hypervisor interaction by the guest can be used to offer contextualization data straight from the Hypervisor itself, using a customized API both to react to changes in context information and to transfer new information. However, this solution requires modifications both to Hypervisor and guest operating system software and would require considerable standardization to be widely available, with regards to the compatibility of virtual hardware APIs between Hypervisor technologies.

Dynamic virtual device mounting is based on dynamically mounting virtual media containing newly generated content in a running VM via the reuse of existing Hypervisor interfaces and procedures [15]. Interoperability is achieved by reusing existing drivers for removable media such as USB disks or CD-ROM drives. Recontextualization can be detected by the guest OS by reacting to events triggered when new USB or CD-ROM media is available.

It is proposed that the dynamic virtual device mounting approach is the most promising solution to recontextualization due to inherent interoperability and support in all ma-

jor operating systems. The ability to manage virtual devices is also offered by the Libvirt API [157], inferring that there is fundamental support for these operations in most major Hypervisors. The following section describes the recontextualization solution in more detail.

6.4.2 A Recontextualization Solution

In this section, an implementation of a system for runtime recontextualization is described, followed by an evaluation to validate the suggested approach in Section 6.5.2. The previously discussed virtual device mounting technique is used in response to migration events and thus enables automatic self-configuration of newly migrated VMs. The following subsections discuss the mechanism and architecture in more detail.

6.4.2.1 Mechanism

Figure 6.11 illustrates the recontextualization approach used in the implementation. Each VM is assigned a virtual CD-ROM device for contextualization on which the host-specific and thus provider contextualization data can be found. When a VM is migrated from one host to another events describing this action are triggered by the Hypervisor, which can be registered to via the Libvirt API. In response to these events, the recontextualizer software triggers a detachment of the virtual device mounted with contextualization information, and once the migration is completed a new virtual device with context information relevant for the new host is automatically attached to the VM as it resumes operation after migration.

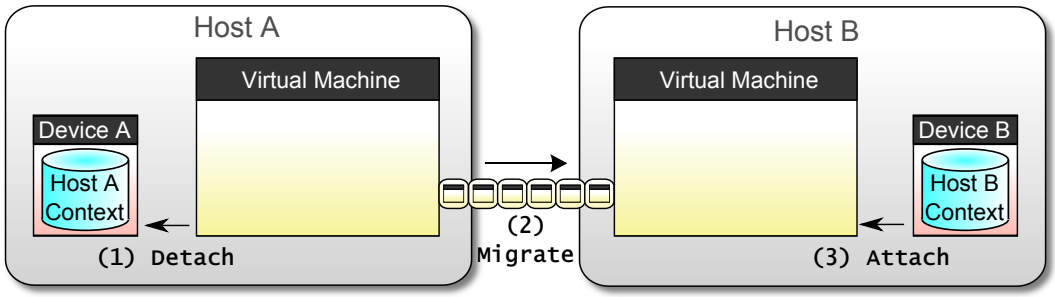


Figure 6.11. Recontextualization approach overview.

Event support including migrations is present in several Hypervisors, including Xen [21] and KVM [149]. The Libvirt API enables a unified approach to VM management available and includes event support. An initial version of the recontextualization

system was implemented using KVM with QEMU [200] specific event and control APIs and the second version was implemented using Libvirt to make the solution Hypervisor independent. Libvirt provides a number of event types that can be monitored via a callback: i) Started, ii) Suspended, iii) Resumed, iv) Stopped, and v) Shutdown. Upon receiving an event callback details are returned on the specific cause of the event, for example the shutting down of a VM on a host machine triggered by migration terminating successfully.

6.4.2.2 Architecture

The architecture of the implemented system is shown in Figure 6.12. Up-to-date context data is dynamically bundled as ISO images on the host. The recontextualizer, implemented in Python, manages the attachment and detachment of virtual CD-ROM devices inside a VM that contain the data held within the ISO image media in response to events from the Hypervisor. The Python Libvirt API bindings were used to access the Libvirtd daemon for the purpose of abstracting the specifics of the underlying Hypervisor and to improve interoperability.

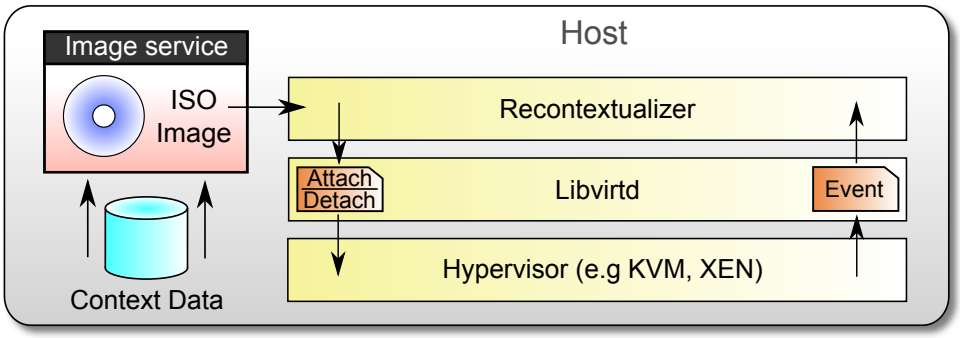


Figure 6.12. Architecture overview.

6.4.3 Detailed Design

The recontextualization mechanism, outlined in Algorithm 1, registers interest with libvirt for VM start and stop events by means of a callback function before beginning its operation. After starting the recontextualization mechanism waits in a loop for new events from Libvirt. When an event is registered in the callback function it is classified and appropriate action taken.

Algorithm 1 Recontextualization Mechanism

```
connection = libvirt.open('URL')                    ▷ Connect to libvirt at a given URL
libvirt.EventRegister(Callback())    ▷ Register interest in event callbacks from libvirt
while true do
    if connection.event then                                  ▷ New event from Libvirt
        function CALLBACK(connection.event)
            if connection.event == vmStop then        ▷ A VM stopped, remove device
                detachDevice(deviceId)        ▷ Force removal of device with given ID
                updateDevices()
            end if
            if connection.event == vmStart then          ▷ A VM started, provide data
                contextData()                              ▷ Generate new context data
                makeIso()                                  ▷ Make ISO CDROM image
                attachDevice(deviceDefinition)      ▷ Attach device specified in XML
                updateDevices()
            end if
        end function
    end if
    Sleep()
end while
```

If a stop event is detected the previous device containing contextualization data is detached and the device change is committed via libvirt, which proceeds to propagate the virtual hardware change at the Hypervisor level. In the case of a start event being detected, new contextualization data is generated then packaged into an ISO CD ROM image. The recontextualization mechanism defines a new CD ROM device pointing to the ISO via a simple XML definition understandable via Libvirt:

```
<disk type='file' device='cdrom'>
  <target dev='hdc'/>
  <source file='/some/iso/file/location/an.iso'/>
  <readonly/>
</disk>
```

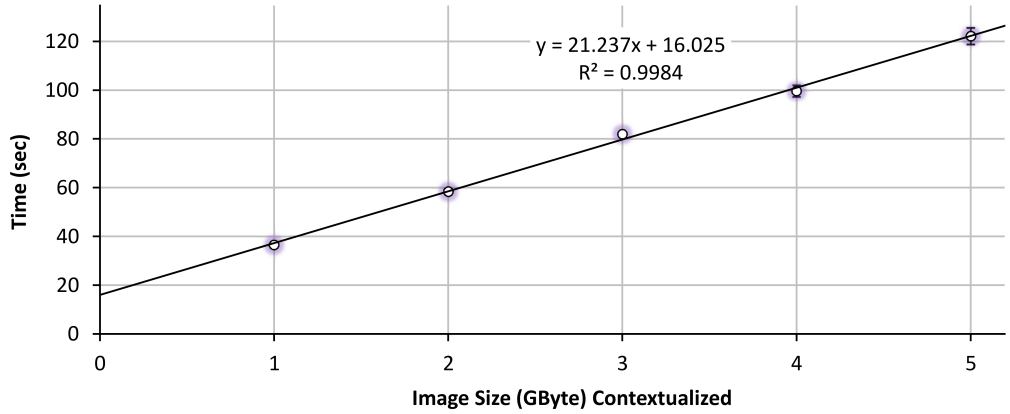
Again the virtual hardware change is then committed to the Hypervisor via a call to the Libvirt API after which the contextulization data used is cleaned up.

6.5 Evaluation

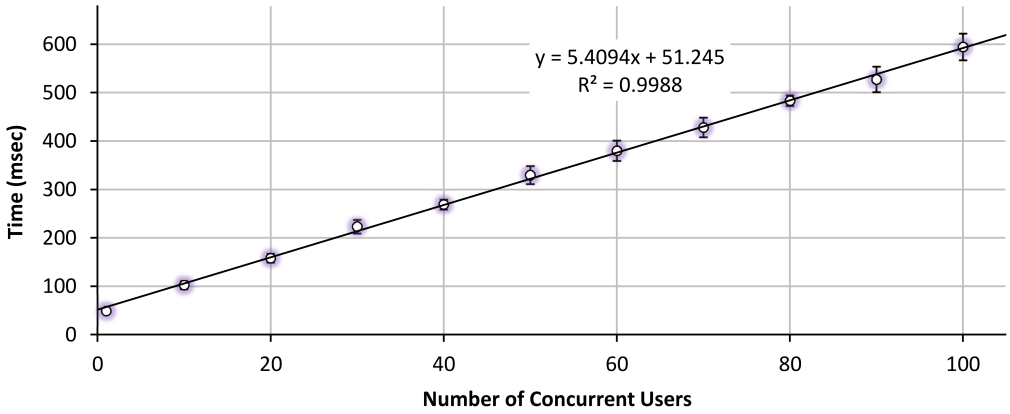
In this section of the chapter the Contextualization Tool performance in a multi-user environment is evaluated including the time taken to covert an image to a specific infrastructure when using value added features such as encryption. The overhead of the Recontextualization approach is assessed using the Hypervisors Xen and KVM and provides an additional insight into the performance each Hypervisor to perform live migration.

6.5.1 Contextualization Tool Performance

To confirm the validity of the contextualisation approach used, an implementation of the VM Contextualizer was tested on the Cloud testbed, outlined in Section 5.4.1, using a Dual CPU (Intel Xeon E5630) server with 16 GB of RAM and 1 TByte WD SATA 7200 rpm HDD. Figure 6.13(a) and 6.13(b) provide evidence on the potential performance of the contextualization approach with regards to preparing a VM Image for receipt of context agnostic of the operating system type used of various sizes in the range of 1-5 GByte in increments of 1 Gbyte and with varying numbers of concurrent user requests from 10-100 in increments of 10, to create ISO CD Images containing 1 Mbyte of context data (See Section 6.3.3.4 for details on the these contextualization steps). The results show adequate scalability and response time over 10 iterations of experimentation with minimal variance, as shown by the error bars on the graphs.



(a)



(b)

Figure 6.13. a) Time to Prepare a VM Image. b) Response Time of Concurrent User Requests to Generate ISO Images.

In addition to the above performance results, the VM Contextualizer is able to convert between image formats with various features enabled, including compression and encryption in the case of QCow2. The performance results are presented in Figure 6.14. Conversion of images is of particular use in more advanced Cloud deployment scenarios such as in Hybrid and Multi-Cloud environments.

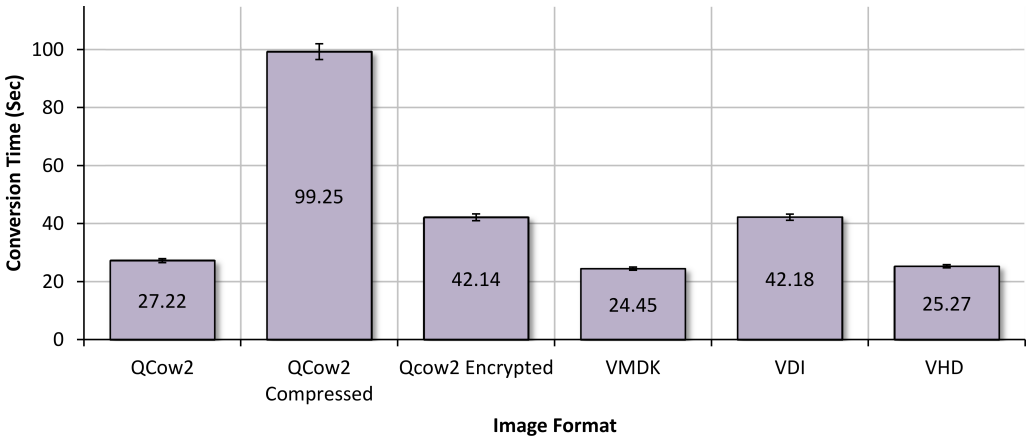


Figure 6.14. Image Conversion Performance Results From Raw.

The conversion experiment was performed on 64Bit Debian 6.0 using a 2.6Ghz Intel Core i7 2677M CPU with 8GB of DDR3 at 1333Mhz and a SandForce SF-2281 SATA III RunCore Sold State Disk. Each conversion was performed over 10 iterations to minimise variance. A image of the Raw format, containing a basic installations of the CentOS 5 Linux operating system with a total size of 1.5GB, was used for conversion. From the results it can be seen that the VM Contextualizer is able to convert between formats in an acceptable time frame given that it can take a few minutes for a VM to boot, taking less than 30 seconds to perform conversion excluding compressed and encrypted images. The results with compression enabled while converting to a QCow2 image illustrate the size benefit of using this feature. After compression the image size was 500MB, a compression ratio of 3:1. Encrypting the QCow2 image during conversion added an overhead of roughly 15 seconds to the conversion process.

The time it takes to convert an image has the potential drawback of reducing the reaction time of a Cloud provider when scaling on demand but the reduction in the time to transfer an image from one provider to another could make conversion with compression a worth while endeavour and provides a interesting trade-off point for future investigation. One must also take into consideration that both compression and encryption have an adverse effect on the performance a VM when carrying out I/O on block storage devices.

6.5.2 Recontextualization Performance

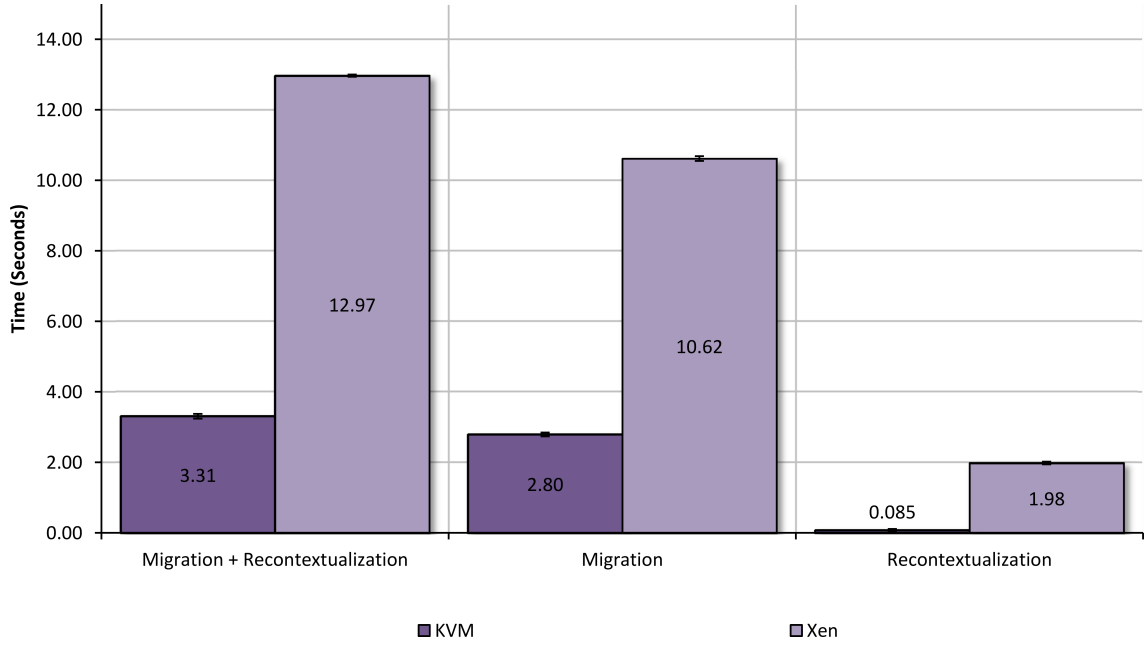
A series of tests to evaluate the feasibility of the recontextualization approach in Section 6.4 have been performed. For all tests, Libvirt version 0.9.9 was used to monitor and manage the VMs. QEMU-KVM version 1.0.50 and Xen version 4.0.0 were used as Hypervisors, both running on the same hardware using CentOS 5.5 (final) with kernel

version 2.6.32.24. The hosts used in these tests are on the same subnet, have shared storage and are comprised of a quad core Intel Xeon X3430 CPU @ 2.40GHz, 4GB DDR3 @ 1333MHz, 1GBit NIC and a 250GB 7200RPM WD RE3 HDD.

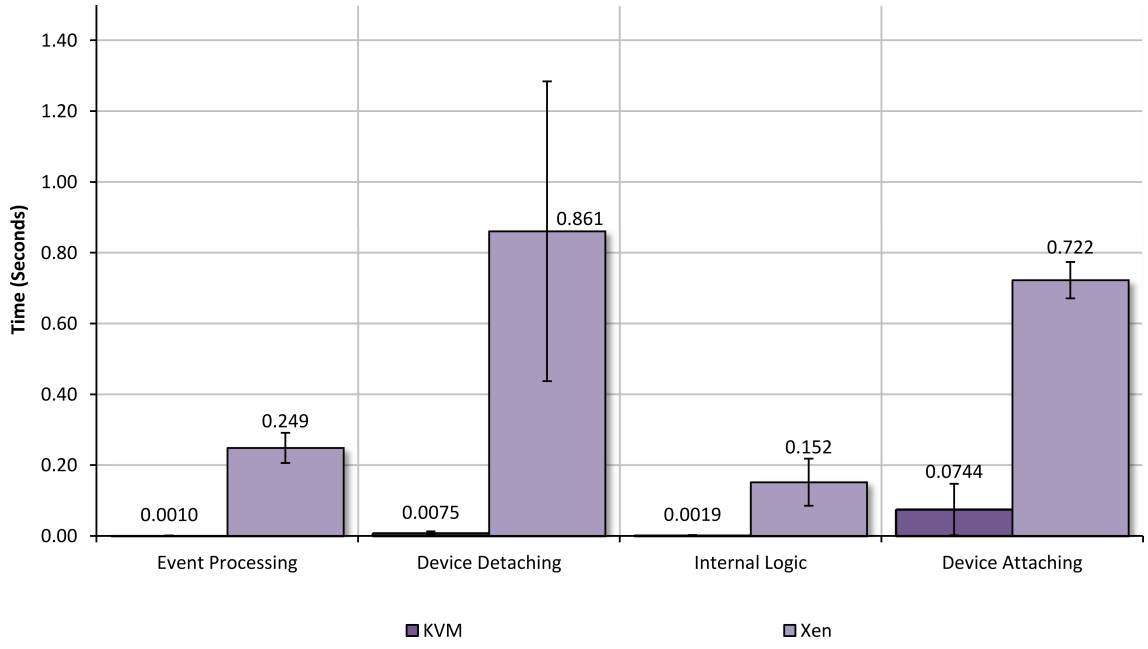
The results of the evaluation are shown in Figure 6.15(a). The first set of bars illustrate the time to migrate a VM from one host to another with recontextualization running and context data attached, and the second set of columns illustrate the same migrations with recontextualization turned off and no virtual devices mounted. The third column illustrates the time spent within the recontextualizer software during the tests from the first column, measured from when the event for migration was received in the recontextualizer until the device had been removed and reattached. The values shown are the averages from ten runs, and all columns have error bars with the (marginal) standard deviations which are all in the 0.03 to 0.07s range.

Based on the evaluation it can be conclude that the recontextualization process adds about an 18% overhead using either Hypervisor compared to doing normal migrations. For KVM, most of the extra time required for recontextualization is spent outside the bounds of the component, likely associated with processing events and extra overhead imposed by preparing migration with virtual devices attached. In the case of Xen the device management functionality in Libvirt proved unreliable and therefore had to bypass the Libvirt API and rely on sub-process calls from the recontextualizer to Xen using the `xm` utility. This workaround increased the time needed for recontextualization in the Xen case.

There are four major phases associated with the recontextualization process. First, information about the VM corresponding to the event is resolved using Libvirt when the migration event is received. In the second phase, any current virtual contextualization device is identified and detached. Third, new contextualization information is prepared and bundled into a virtual device (ISO9660) image. Finally, the new virtual device is attached to the VM.



(a)



(b)

Figure 6.15. a) Time Measurements of Recontextualization. b) Breakdown of Time Spent During Recontextualization.

A detailed breakdown of the time spent in different phases of recontextualization is presented in Figure 6.15(b). The above mentioned workaround for Xen interactions affects the second and fourth phase (detaching and attaching of devices), most likely increasing the time required for processing. In the first and third phases Xen requires sig-

nificantly longer time than KVM despite the VMs being managed using the same calls in the Libvirt API, indicating performance flaws either in the link between Libvirt and Xen or in the core of Xen itself.

6.5.3 Feature Comparison

Due to the inherent novelty of the Contextualization and Recontextualization mechanism presented in the Contextualization Tool in this Chapter, there is little in the way of related technology that provide similar approaches, features and functionality. For what technologies that do exist and exhibit some form of contextualization, the following table provides a summary of their differences:

Table 6.3. Comparison of Technologies with Similar Functionality to the Contextualization Tools

Feature	**Contextualization Tools [Chapter 6]**	**OpenNebula [188]**	**Nimbus [177]**	**Contrail [52]**
Infrastructure Level Contextualization	Supported	Supported	Supported	Supported
Platform Level Contextualization	Supported	Unsupported	Unsupported	Unsupported
Recontextualization On Migration	Supported	Unsupported	Unsupported	Unsupported
ISO As Context Data Storage Medium	Supported	Supported	Unsupported	Supported

6.6 Summary

In this chapter issues surrounding configuring the Cloud service software stack were discussed including the relevance of Autonomic computing and Configuration Management tools in combating the management headache of the Cloud's large scale. Three configuration management tools were compared and their suitability in Cloud Computing assessed. A conclusion was draw that although these tools are useful management aids, they do not accommodate for the dynamic nature of Cloud environments or cover the configuration of the entire Cloud service software stack.

The nature of contextualization was introduced in the light of the OPTIMIS Project and details presented on the implementation of a tool for the contextualization of platform level services as well as virtual infrastructure that can be deployed alongside a range of PaaS and IaaS solutions due to its generic self-contained architecture. The implication of contextualization in Clouds was discussed, the motivation behind the research and a landscape suggested for the evolution of contextualization tools across all classes of Clouds as part of the ecosystem of the future. Both the image and instance level contextualization of VMs was contributed and the potential effectiveness of the Contextualization Tool illustrated through an example scenario and performance results. The need for contextualization of platform tools in the OPTIMIS toolkit was described and the problems overcome to support it. Finally, some related work on alternative contextualization mechanisms were discussed and differentiated from the work presented in this chapter.

Furthermore, the chapter discussed the data gathered by the Contextualization Tool and how this is achieved through a number of interactions with devices and services within a VM's environment. This enables updates to be made to a Cloud application's entire software stack and is only made possible via the proposed contextualization approach's support for the configuration of platform level services. This ability is a major distinguishing feature of the tool from other approaches in the literature.

In addition, recontextualization: the autonomous updating of configuration during runtime was described and defined. Moreover, it was shown that recontextualization is a key enabler to using multiple Cloud sites concurrently. Different alternative mechanisms were evaluated for recontextualization based on a set of requirements. The chosen approach, based on automatic mounting of dynamically generated images as virtual devices, is highly interoperable supporting a variety of Hypervisors and virtually all operating systems. Apart from CD-ROM mounting routines, which are standard in most operating systems, no custom software is required inside the guest VM to make the contextualization data available at the file system level.

Chapter 7

Cloud Resources for Research

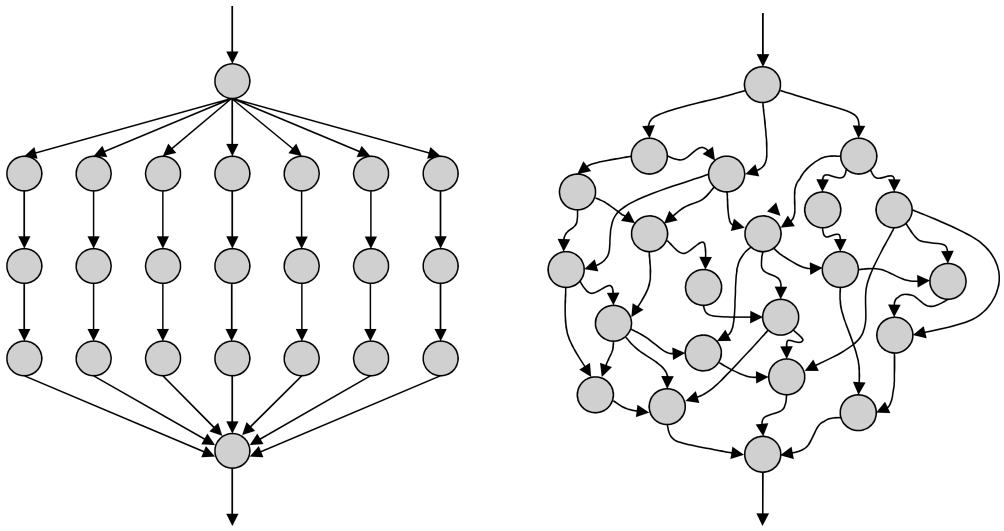
7.1 Introduction

With the advent of pay per use on demand Cloud Computing, the academic community has shown an interest in adopting the paradigm to facilitate and speed up research [14, 23, 57]. Clouds offer many functional and economical benefits to academic users including a close fit with the short life-cycle of research projects where costs savings can be made by renting infrastructure rather than incurring large hardware procurement costs. In addition Clouds provide a reduction in resource contention where the quantity and size of Cloud resources can be tailored and provisioned to fit a user's application requirements. An academic need not wait for shared resources, as is typical in Grids, with resources provisioned on demand. Finally, reductions in operating expenditure are achievable through the economy of scale enabled by a Cloud provider and by outsourcing the costs of infrastructure support staff.

On the other hand, there are a number of drawbacks and hurdles to overcome before academic insinuations fully adopt Cloud Computing. Clouds evolved to support SOA and scalable web based applications that do not fit well with the historical design of scientific applications that achieve scalability and parallelism through batch oriented middleware such as The Globus Toolkit [88], UNICORE [218, 250] and gLite [154] in addition to other technologies the likes of the Message Passing Interface (MPI) and OpenMP [187]. Typically these applications tend to be short-lived and workflow based, while Clouds

applications often persist over many months or years.

Figure 7.1 illustrates typical workflow types run on Grid infrastructure [161, 284], the dependencies between jobs and are classified as Balanced and Unbalanced. Balanced workflows consist of parallel chains of tasks (pipelines) connected with an initial and terminal node, as shown in Figure 7.1(a). The number of nodes in each chain is the same. Unbalanced workflows are arbitrary acyclic graphs, also with initial and terminal nodes, as shown in Figure 7.1(b). Both workflow types pose different challenges when mapping scientific applications to Cloud resources. Balanced workflow pipelines provide a reasonable fit with the resource spawning mechanism found in Clouds, where VM of the same type are duplicated from a based image and would enable resources currently unused (such as the initial node) to be terminated or the state of a VM saved to disk for later use. The chaotic nature of dependencies in Unbalanced workflows is more problematic and could possibly limit efficient Cloud resource usage.



(a) (b)

Figure 7.1. Typical Workflow Structures: a) Balanced b) Unbalanced.

Another hurdle to overcome is the dynamic nature of Cloud environments where virtual machines are brought on and offline frequently as needed and as demand changes. This is in contrast to the relatively static resource pools of Grids, meaning that current scientific applications are not designed to fully leverage the benefits of VM elasticity. In addition, many scientific applications require HPC hardware such as Infiniband or have large memory footprints in the range of 128-512GB of RAM which is above and beyond what can typically be found in Clouds comprised of commodity servers. This is slowly being rectified with Amazon introducing HPC instances in limited quantities [6].

In the previous chapter contextualization was introduced as a concept for abstracting away the problems surrounding the use of dynamically changing resources at the software and platform level. This chapter discusses its use and application in an academic context where a static Grid environment is augmented with contextualization and executable on an IaaS provider's virtual resources with the added benefit of provider neutrality and interoperability above and beyond the current state of the art solutions. The concept of virtual Grids is introduced and an implementation presented with its effectiveness highlighted by the research it has enabled in the School of Computing at the University of Leeds.

In addition, the impact of Cloud technology on legacy applications used in academic research is discussed and light shone on how contextualization can facilitate in migrating these applications to the Cloud. An architecture is proposed for enhancing the scalability of a legacy pathology application using contextualization. Results outline performance and scalability bottlenecks that could be addressed by the proposed rearchitected pathology application. Finally a model of demand is presented for testing the performance and scalability of legacy applications in the Cloud.

The contributions of this chapter are as follows:

- A software implementation of an interoperable self configuring virtual grid infrastructure. Deploying grid infrastructure onto Clouds in an self-configurable and provider interoperable fashion is a key step to providing support for legacy applications. Consolidating the differences between the static environment of traditional grids and the dynamic environment of Cloud infrastructure is challenging. The ability to add resources to a grid middleware dynamically requires careful orchestration and a deep understanding of Grid resource management is needed to enable the development of a software solution that can support a multitude of Grid applications.

- Insights on re-engineering a legacy pathology application deployable on a Cloud with improved scalability as there is little understanding of the implications and benefits that Cloud Computing can offer for the deployment of legacy applications. Porting the pathology application is non-trival as the Cloud architecture to support it is an order of magnitude more complex than the application itself and requires bespoke contextualization. In addition, to the performance of the Cloud enhanced application, a non-trivial model of demand is required for testing in a realistic fashion. Finally, contextualization of a legacy application in not trivial because of the high number and complex interaction of platform level components.

This rest of this chapter is organised as follows: Section 7.2 introduces the standards used and components of the Globus Toolkit. In addition, the concept and implementation

of a Virtual Grid, runnable on Cloud infrastructure, is presented. Section 7.3 discuss the migration of legacy applications to Clouds, proposes an architecture for a Cloud enhanced legacy pathology application and defines a model for placing demand on Cloud resources. Finally, Section 7.4 evaluates the features of the implemented Virtual Grid, the Globus Virtual Cluster, against functionality provided in Nimbus for creating "*one-click-clusters*" and presents performance results on the pathology application.

7.2 Globus Middleware in the Cloud

The Globus Toolkit [78, 88] has become the *de facto* standard in Grid middleware and due to its popularity is thus the focus of the research. This does not mean that the results presented here in are not applicable to other middlewares as, discussed in Section 4.5. For example, the contextualisation approach discussed later in the next subsections could be applied elsewhere because of the features and functionality shared between middlewares. The toolkit enables the selective use of some or all of its components to support specific types of grid application. This can range from simple parameter sweeps, to complex workflows and in in some versions SOAs.

The Globus Toolkit implements a number of standards created by the OGF [183] including: Open Grid Services Architecture (OGSA) based on the work of Foster et al. [77] and OGSI [248] intended to provide an infrastructure layer to OGSA that was developed to account for the stateless nature of Web Services (WS) and extends them to accommodate for the transient and stateful nature of the Grid. This functionality is now obsolete and instead replaced by WSRF [276]. WSRF provides a set of operations that web services may implement to become stateful where clients can communicate with services that allow data to be stored and retrieved.

The toolkit is secured via an implementations of the Grid Security Infrastructure specification. This includes certificate based authentication and identification tools. To stage data into and out of Grid resources, Globus makes use of the GridFTP [4] protocol for its data management needs. GridFTP is an encryptable, high performance implementation of the original FTP that makes use of multiple TCP connections to saturate network connectivity to obtain maximum bandwidth.

In the Globus Toolkit, resources are managed via the Grid Resource Allocation and Management Protocol (GRAM) [99], a set of services that supports communication with a number of job schedulers and batch queuing systems, such as the Portable Batch System (PBS) [196], via a common protocol. GRAM can locate, submit, monitor and cancel jobs providing reliable operation in the case of resource failure achieved with stateful

monitoring. Moreover, GRAM can operate securely with the use of GSI for credential management and uses GridFTP for file staging. Monitoring of resources is achieved via the Monitoring and Discovery Service (MDS) [100]. MDS enables resources to be registered along with associated properties and attributes and is presented as an index service that can be queried by clients for information on available resources.

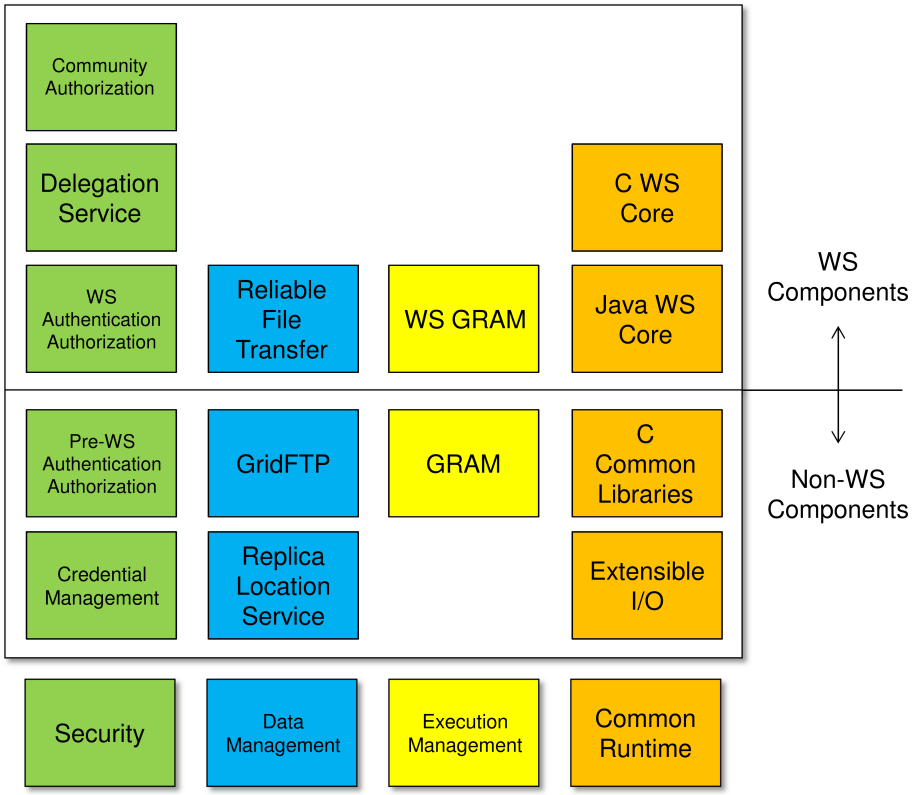


Figure 7.2. Components in Globus Toolkit Version 4.

The Globus Toolkit has evolved over multiple iterations of development, implementing new features and adopting Web Services and SOA. Version 3, GT3 [109], of the toolkit was the first to introduce Java web service technology as a means to offer an application-to-application integration framework for the creation of Grid services. The WSDL [275] was adopted as standard interface and binding description language. This version of the toolkit was poorly received by the academic community due to performance and stability issues caused by a complete rewrite of many components in the toolkit to support Web Services. There was also a lack of interest in the adoption of SOA design from the science community, who are adversed to change and preferred to continue to use their existing knowledge and expertise in developing job and batch oriented applications.

GT4 [78, 79, 110] brought improvements to performance and scalability with the introduction of GRAM4 [102] but the complexity of the toolkit, as illustrated in Figure 7.2,

continued to cause headaches when setting up scalable Globus enabled clusters. Version 5 of the toolkit, GT5 [111] as depicted in Figure 7.3, rectified this issue with the introduction of GRAM5 [103], an enhanced version of GRAM2 [101] that removed the Web Service and SOA aspects of the toolkit that were poorly received by the academic community. In essence the latest version has regressed to GT2 [108], with the improvements from GT4, simplifying maintenance. The Java WS Core, Reliable File Transfer component, WS enabled GRAM4 and the programming model used in GT4 have instead migrated to a new project named Globus Crux [90].

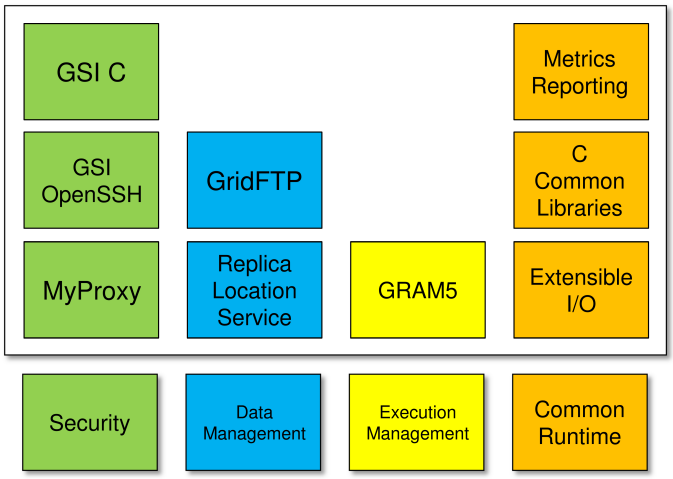


Figure 7.3. Architectural Overview of the Globus Toolkit Version 5

7.2.1 Virtual Grids

The concept of a Virtual Grid pertains to the the execution of Grid middleware on top of Cloud infrastructure which in turn runs Grid enabled applications. Nimbus [177] is an example that provides virtual workspaces of the Globus Toolkit for application development that arose due to the difficulties associated with configuring the toolkit. Due to the dynamic nature of Cloud Computing and its inherent scalability enabled through virtual machine elasticity, configuring Grids to function on top is a non-trivial task. This is complicated further by the multitude of job submission mechanisms and underlying programming models supported in Grid environments. Figure 7.4 illustrates an example software stack necessary to run a workflow based image processing application utilizing both Cloud IaaS and Grid middleware.

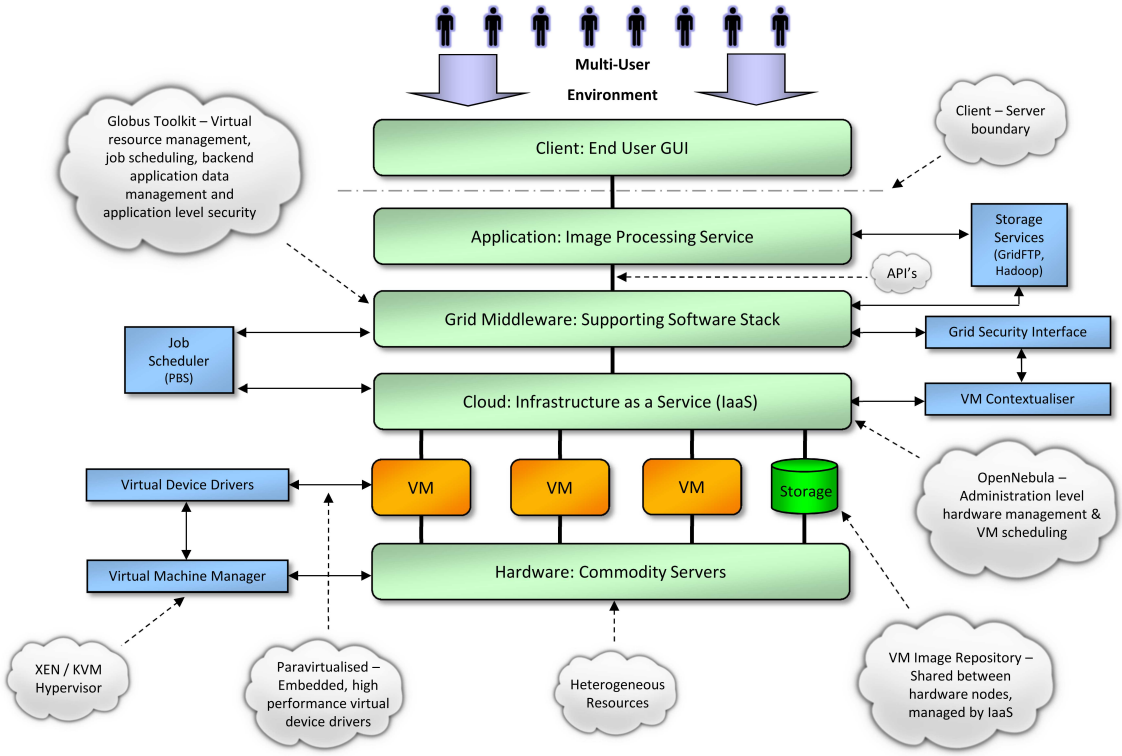


Figure 7.4. Virtual Grid Architecture.

From the figure it can be seen that the use of a Virtual Infrastructure Manager provides a mechanism to control a virtual cluster of Grid resources built on top of on a single instantiated base image. A base image is needed that can be contextualized in such a fashion as to enable the dynamic removal and addition of virtual Cloud resources without interfering with the operation of the application running on top of the Grid middleware. Contextualization needs to account for security, enabling explicit white list certificate based access to the virtual resources. In addition, resource management at the Grid level by whatever job scheduler or batch queue management system is used, must be contextualized in such a fashion as to be able to deal with an unreliable cluster of resources. Finally, any base image must be instantiated and contextualized such that it is agnostic of the underlying Cloud infrastructure. This enables interoperability amongst different Cloud providers and would allow academic institutions to avoid vendor lock-in and combat the technological fracturing found in the Cloud ecosystem.

7.2.2 An Implementation - Globus Virtual Cluster

To investigate the possibility of running Grids on Clouds in an interoperable fashion above and beyond the current state of the art, a Virtual Grid was implemented using the Globus

Toolkit at its core. The middleware was configured to enable the execution of Web Services (via Java WS Core containers), job and batch applications via the integration of GRAM4 with Torque [247]. Torque is an actively developed open source resource manager forked from PBS that provides control of batch jobs over distributed compute nodes. Torque provides additional features and functionality that enhance scalability and reliability. Torque was configured to use Maui [164], an open source cluster scheduler supporting a number of scheduling polices including advanced reservation and fairshare. Support for secure data management and authorization/authentication was enabled via the use of GridFTP and GSI respectively. In addition, MPI job submission was enabled by installing and configuring MPICH2 [215] with a Torque enabled `mpiexec` binary avoiding the need to configure `mpd`, the MPI daemon, on worker nodes, which complicates MPI job submission. The architecture of the Globus Virtual Cluster can be seen in Figure 7.5

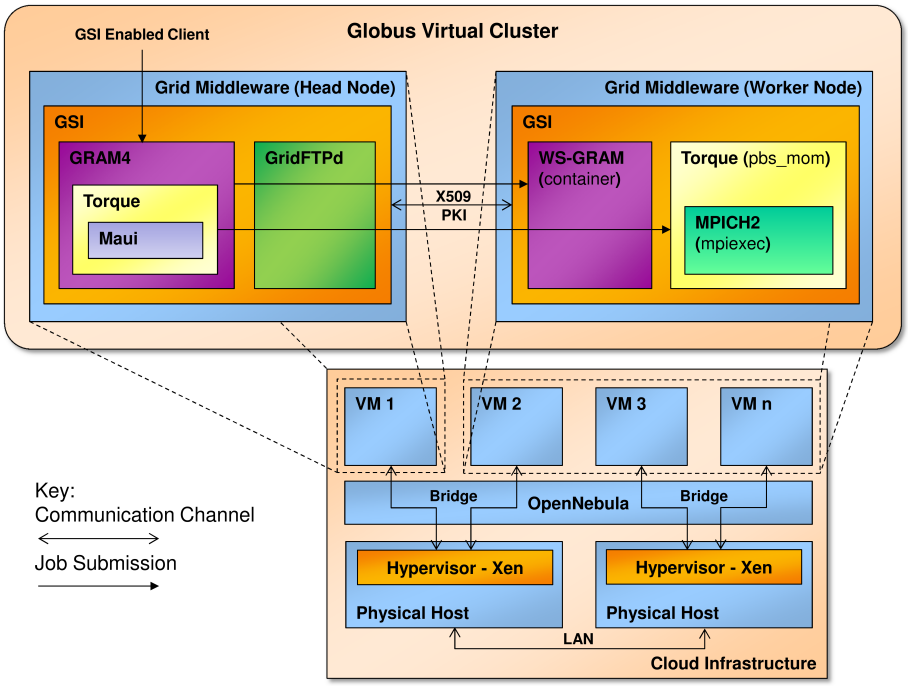


Figure 7.5. Globus Virtual Cluster Architecture.

No components in the Globus Virtual Cluster interact with the underlying Cloud infrastructure. The Grid middleware is thus unaware that it is running virtualised across an number of virtual machines in a dynamically changing environment. This enables interoperability with a wide number of IaaS provides as long as the base virtual machine image, with which the Grid middleware software stack is installed in, is of a format that is supported by the IaaS provider's Hypervisor.

The Globus Virtual Cluster makes use of contextualization, as outlined in Chapter 6,

to set the networking context in either dynamic DHCP or static environments and selects an appropriate pre-generated client and server certificate at boot time enabling the contextualized node to be accessible by other nodes in the cluster. The first node to come on-line automatically takes on the role of the head node, maintains a list of available worker nodes and acts as a gateway to GRAM via a "globus" client account accessible via GSI OpenSSH that uses X509 certificates and the OpenCA [186] Public Key Infrastructure (PKI).

Fault tolerance and reliability mechanisms in the Grid middleware software stack are configured so that the cluster can take advantage of new virtual resources that come online and also accommodate for VMs that are taken offline for cost savings when enhanced scalability and elasticity is no longer required. Jobs that fail due to a resource terminated at the Cloud infrastructure level are reassigned to nodes that are currently online and re-executed when these nodes become available.

There are a number of limitations with the current implementation. Since certificates are pre-generated for use with GSI there is a predefined upper limit on the number of virtual resources that can be brought online. This is currently statically configured to 1000 nodes and can be increased if needed. In addition, there is no monitoring of the Grid middleware performed by the Cloud IaaS to enable the dynamic scaling of resources on demand given a certain KPI such as the queue depth of the Maui scheduler. This is planned future work but the user who created the cluster is able to pro-actively provision more resources manually to reduce the runtime of an application.

A number of tests were created to confirm the correct configuration of all the components within the software stack of the Globus Virtual Cluster due to the complexity of the system and to facilitate interoperability testing on IaaS providers. To validate that all the components within the Globus Toolkit were configured correctly, two tests were created that submitted dummy jobs using both WS enabled GRAM and traditional Pre-WS GRAM. The WS-GRAM test utilised the Resource Specification Language (RSL) [214] to define a job, while the Pre-WS GRAM test used the `gatekeeper` daemon to dispatch a job using the GSI library for communication.

Listing 7.1. GRAM4 Test RSL Job Description

```
<job>
  <executable>my_echo</executable>
  <directory>/home/globusclient/ws-gram-test</directory>
  <argument>Hello</argument>
  <argument>World!</argument>
  <stdout>/home/globusclient/ws-gram-test/stdout</stdout>
  <stderr>/home/globusclient/ws-gram-test/stderr</stderr>
  <fileStageIn>
    <transfer>
     <sourceUrl>gsiftp://globus01:2811/bin/echo</sourceUrl>
     <destinationUrl>file:///home/globusclient/ws-gram-test
         /my_echo</destinationUrl>
    </transfer>
  </fileStageIn>
  <fileCleanUp>
    <deletion>
     <file>file:///home/globusclient/ws-gram-test/my_echo</
         file>
    </deletion>
  </fileCleanUp>
</job>
```

As part of the WS-GRAM test, the use of GridFTP was specified in the RSL for staging data to worker nodes to confirm that GSI secured GridFTP (GSIFTP) was fully operational and is shown in Listing 7.1. An additional test was created to confirm that the execution of MPI jobs was possible. This test utilised GRAM2 RSL and the underlying "`qsub`" command to test Torque and Maui functionality. The RSL used in this test is shown in Listing 7.2.

Listing 7.2. Job Description for Testing MPI with GRAM2 RSL

```
&(jobtype=mpi)(executable='/home/globusclient/mpi-test/
    helloworld.bin'(count='8'')
('stdout='/home/globusclient/mpi-test/stdout.txt')(stderr='/
    home/globusclient/mpi-test/stderr.txt')
```

An unforeseen benefit of the development of the Globus Virtual Cluster, in addition to evaluating whether running Grids on Clouds in an interoperable fashion is possible, has

been the facilitation of research in the subject area of Grid Computing at the University of Leeds, School of Computing by a number of postgraduate students [55, 224]. These students have made use of the Globus Virtual Cluster to perform experiments on the renegotiation of SLA and to gain an understanding of Web Services in Grid architectures. In addition, a number of final year undergraduate students have performed quantitative evaluations on the negligible performance overheads of running MPI jobs on virtualised Grid infrastructure.

7.3 Legacy Applications in the Cloud

Many academic institutions and businesses depend on applications written before the advent of Cloud Computing for day to day operations. Legacy systems tend to run on expensive obsolete platforms and are often written in archaic programming languages that are not well understood by current staff. In the previous chapter contextualization was discussed as a mechanism to enable the configuration of applications deployable onto Cloud infrastructure. In this subsection contextualization enables the simplified migration of legacy applications to the Cloud by providing a framework with which to integrate with that abstracts away the dynamic nature of the Cloud while enabling enhanced performance through scalability.

Legacy applications have been defined as:

Definition *Legacy Application* Any information system that significantly resists modification and evolution. [33]

Legacy applications are often used well beyond their intended lifespan and can continue to have an impact on an organization even after decommissioning. Their continued use can be for a variety of reasons such as to gain a return on an investment or because of cultural issues associated with change management and result in maintenance challenges. In addition, there is motivation to migrate away or further enhance integration with legacy applications to improve usability and performance respectively.

Migrating legacy applications away from the underlying hardware with which they are run was not possible before the arrival of advanced Virtualization techniques. As discussed in Chapter 5, Virtualization enables the emulation of entire instructions sets via binary translation providing a layer of abstraction that can be used to deploy a legacy application unaltered on to newer but dissimilar hardware. This provides the benefit of maintaining the stability of the application and removes the expense of hiring experts

to port legacy code to a new architecture. Cloud Computing presents additional opportunities to enhance legacy applications by enabling improved scalability in combination with Virtualization and by enabling the outsourcing of resources reducing the total cost of ownership.

There are performance ramifications for migrated legacy applications that are not from the x86_64 architecture typically found in Clouds. Many enterprise legacy applications run on RISC processors that are big endian and based on the Power and SPARC architectures. This is not as much of a potential performance issue as it would seem as most applications running on expensive legacy servers are running in resource partitions and have resources requirements that are usually smaller than the entire server with which they run on or the resource utilization rate is near the data center norm of 15%. In addition, the performance difference between modern processors and legacy servers is such that an application running in a legacy server can easily be accommodated by a modern Cloud server.

There are different ways in which legacy applications can be migrated that deal with a number of architectural issues [26]. In Clouds, to provide the benefits of enhanced scalability, legacy applications can be encapsulated to enable API interoperability and then combined with existing technologies, including Cloud platform services such as block storage and load balancers. The rational behind this approach is to maximise the utility of technologies already designed to scale on the Cloud. The remainder of this section provides details on this approach in the context of a legacy Pathology application that exhibits poor scalability as its user base increases.

7.3.1 A Cloud Enhanced Pathology Application?

To analyse the ramifications of migrating a legacy academic application to the Cloud, a Pathology application that services requests for slide images was chosen as a suitable candidate as its architecture exhibits poor scalability when accessed by many concurrent users. These performance related issues are discussed in detail within Section 7.4.2.

The Pathology application provides the functionality of a digital microscope that aids pathologist in the analysis and visualisation of gigapixel tissue slides as illustrated in Figure 7.6. It enables greater efficiency when diagnosing diseases at a cellar level by allowing a pathologist to browse and analyse slides via a client. In addition, analyse can be performed automatically via the use of image processing algorithms, an ongoing area of research at the School of Computing at the University of Leeds [43]. An example of such an algorithm is Nuclear Quantification, where the number of cell nuclei in a given

area of tissue can be used as an indicator of diseases such as cancer. The application is makes us of the commercial software offerings of Aperio [11], a specialist in solutions for digital pathology. The specific product that was enhanced is the Aperio ImageServer [128] that services requests for parts of a slide.

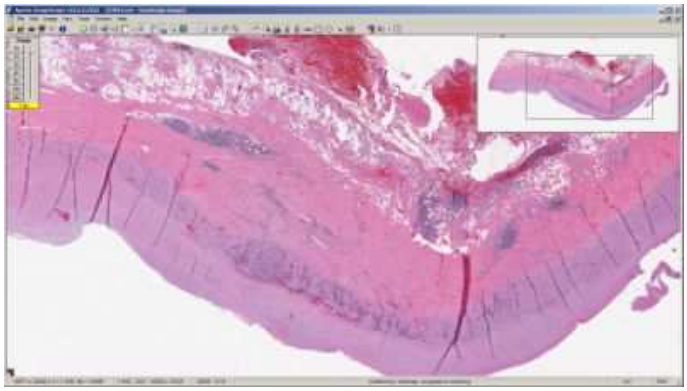


Figure 7.6. Visualised Pathology Slide Served By Aperio ImageServer.

Aperio ImageServer provides a high-performance interface to digital slide images stored as SVS files, a proprietary `TIFF` format that utilize `JPEG2000` compression. The software acts as a web server making it a perfect candidate for "*cloudification*" given the web oriented nature of Cloud technologies. The web server returns designated regions of an image using a HTTP based API. Image sizes up to 250,000 pixels in width and height are supported as well as server-side algorithm processing.

The architecture of the ImageServer is based on a monolithic client-server design where a single process services all requests. Under heavy load or during periods of heavy traffic, such as flash crowd events where a surge of users access the server, the software exhibits very high response times. In the following subsection an architecture is proposed to address this issue using the principles of Cloud Computing.

7.3.2 Proposed Architecture

The ImageServer places both high load on computational and data resources due to algorithmic analysis performable and the size of the slides with which are operated on. This has consequences for any architecture aiming to improve scalability and brings the requirement that scalability must be enhanced for both compute and data resources in the Cloud. Figure 7.7 illustrates the proposed Cloud architecture for the Pathology ImageServer application. The architecture is comprised of three tiers:

- Client Layer

- Application Layer
- Data Layer

Each tier of the architecture is made up of a set of cooperating VMs. Each set of VMs is fabricated from a single base VM image and is assigned context at runtime, through a set of contextualisation scripts embedded in the image as outlined in Section 6.3.

A Client Layer is included for the purpose of performance testing, emulates the workload of a pathology application using "`wget`" and the Load Generator component. The "`wget`" software package uses HTTP calls to interface with the Application Layer, via the Aperio Web API, to retrieve sub-images of slides held in the Data Layer. The Load Generator, embedded in a VM Head node, utilises a global usage trace for the entire system and generates individual traces from it for each VM. The individual traces define the frequency of requests that particular VM should make to the Application Layer over the lifetime of the VM. The Load Generator, using the global usage trace, controls the number of Client VMs that are online via the Contextualizer component and provides context to the VMs in regards to what specific trace the VM will perform. The Contextualizer has a mechanism to alter the number of online Clients VMs through an interface to the underlying Virtual Infrastructure Manager (VIM).

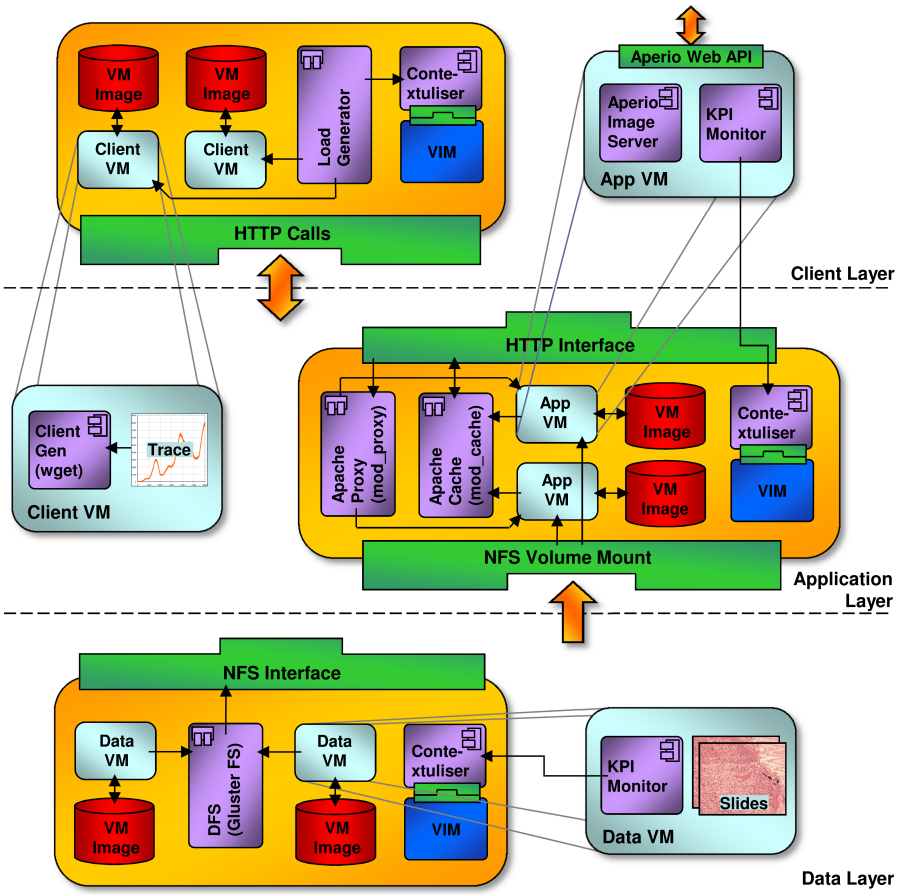


Figure 7.7. Three-tier Architecture of Cloud Pathology Application.

The Application Layer VM instances contain the Aperio Image Server binary and a component for monitoring KPIs. In addition to the VMs, an installation of the Apache HTTP Server is embedded in a VM head node and is used to cache and proxy requests. The Apache Cache stores requests containing frequently retrieved slide data bypassing the Application VMs and the retrieval overhead associated with accessing the Data Layer. When a cache miss occurs, the request is passed to the Apache Proxy, which is used to load balance requests from the Client Layer across multiple Application VMs, distributing the workload to a single instance of the Aperio Image Server. This instance retrieves slide data from a mounted NFS volume, which is then returned to the Apache Proxy. The Apache Proxy returns the data to Application Layer by mapping the serviced request back to the specific Application VM that made it. Concurrently the data is cached for future use in the Apache Cache. The KPI Monitor is used to monitor the resource usage and average request response time of the Aperio Image Server instances. This data is provided to and digested by the Contextualiser component, which caters for changes in demand by provisioning the correct number of VMs, at the VIM level, as the need arises.

The Data Layer provides a DFS over NFS via a set of Data VMs that provide virtual RAID to the Application Layer through GlusterFS [91]. GlusterFS was selected over other DFSs such as Hadoop HDFS [113, 270] due to support for mounting files systems at the operating system level, a requirement for legacy applications that cannot be altered. Virtual RAID 0 or mirroring of data is used to increase I/O throughput, where by each Data VM contains a complete set of slide files. Virtual RAID 0 is justified due to the embarrassingly parallel nature of the client requests. I/O operation per second are monitored by the KPI Monitor and sent to the Contextualiser component, which alters the number of Data VMs executing on demand and situates each VM image on a unique HDD resource to prevent contention.

7.3.3 Modelling Demand

To test the design of the Cloud enhanced Pathology architecture, planned future work discussed in Section 8.3, a model of the demand on the system is necessary to demonstrate scalability in the two dimensions:

- Computation
- Data

Although there are Cloud simulators available, such as CloudSim [38], there is a lack of realistic workload generators that emulate real Cloud usage patterns. In addition, since this simulation tool is based on GridSim [106], workload traces available for CloudSim are based on parallel, Grid and supercomputing systems that do not fit the on demand service provisioning paradigm of Cloud Computing. Current CloudSim users must resort to implementing their own workload generators.

As data on the usage of real Cloud applications is hard to come by due to the closed nature of Cloud hosting environments where provider's operate behind closed doors, a formal model has been defined to create a sensible approximation of a workload of the Pathology application in a real Cloud environment. Requests are generated using a Poisson distribution, as for the sake of demonstration this gives a good approximation for request inter-arrival rates within service based systems [137]. There is however controversy surrounding the use of the Poisson distribution in the modelling of computer networks [195]. In Figure 7.8 when an event ε occurs, it is defined by three phases:

- Ramp Up
- Sustained

- Ramp Down

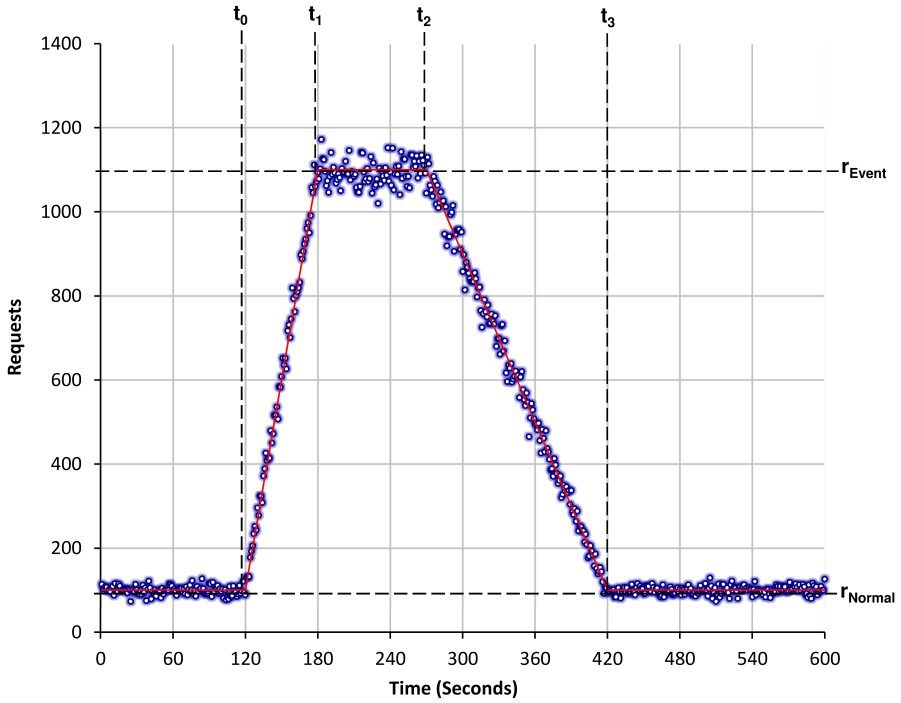


Figure 7.8. ScalabilityTime.eps

The Ramp Up phase α occurs between t_0 and t_1:

$$\alpha = t_1 - t_0 \tag{7.1}$$

In this phase, a linear increase in load on the system is experienced. The rate of the Ramp Up phase is defined by:

$$\Delta\alpha = \frac{\alpha}{r_{Event} - r_{Normal}} \tag{7.2}$$

The Sustained Phase β occurs between t_1 and t_2:

$$\beta = t_2 - t_1 \tag{7.3}$$

Finally, the Ramp Down phase γ occurs between t_2 and t_3:

$$\gamma = t_2 - t_1 \tag{7.4}$$

The rate of the Ramp Down phase is defined by:

$$\Delta\gamma = \frac{\gamma}{r_{Event} - r_{Normal}} \tag{7.5}$$

Altering the ratio of time between the three phases $\alpha : \beta : \gamma$ over the duration of an event ε, where $\varepsilon = \alpha + \beta + \gamma$, will provide an insight into the system and its ability to react to change. Figure 7.9 illustrates how the changes in demand defined by the model will cause the system to react, starting and terminating VMs from the perspective of the application layer. Once the KPI reaches a certain threshold additional resources are provisioned to cover the increase in load. Each VM has a specific number of requests that it can service without adversely effecting QoS and the response time of a request.

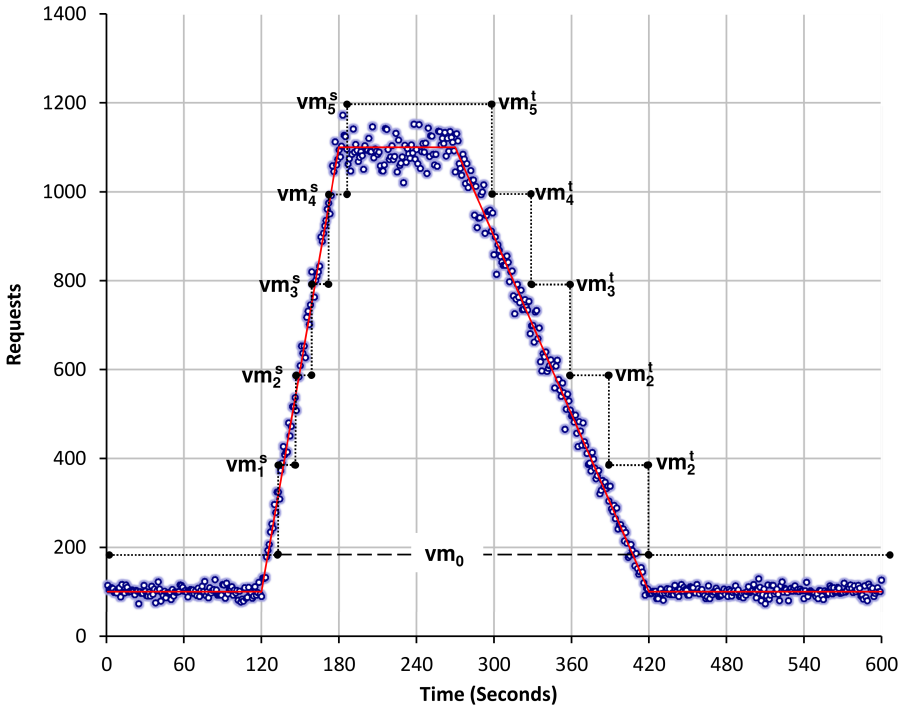


Figure 7.9. On Demand Scaling of VMs

In the simplest case where vm_0 is sufficient to cope with demand outside of ε the total number of VMs required for ε can be calculated by:

$$n = \frac{r_{Event}}{\sigma} \tag{7.6}$$

Where r_{Event} number of requests are serviced by provisioning n number of identical VMs with the property σ, the maximum number of requests a VM can service without breaching a given SLA. Given n, the total capacity σ_{max} of all VMs is:

$$\sigma_{max} = \sum_{i=1}^{n} \sigma_i \tag{7.7}$$

Due to the time overhead, shown in Figure 7.10, of bringing a new virtual resources online, interesting opportunities existing in exploring prediction methods for when best to scale and will be discussed later in Section 8.3 as part of future work.

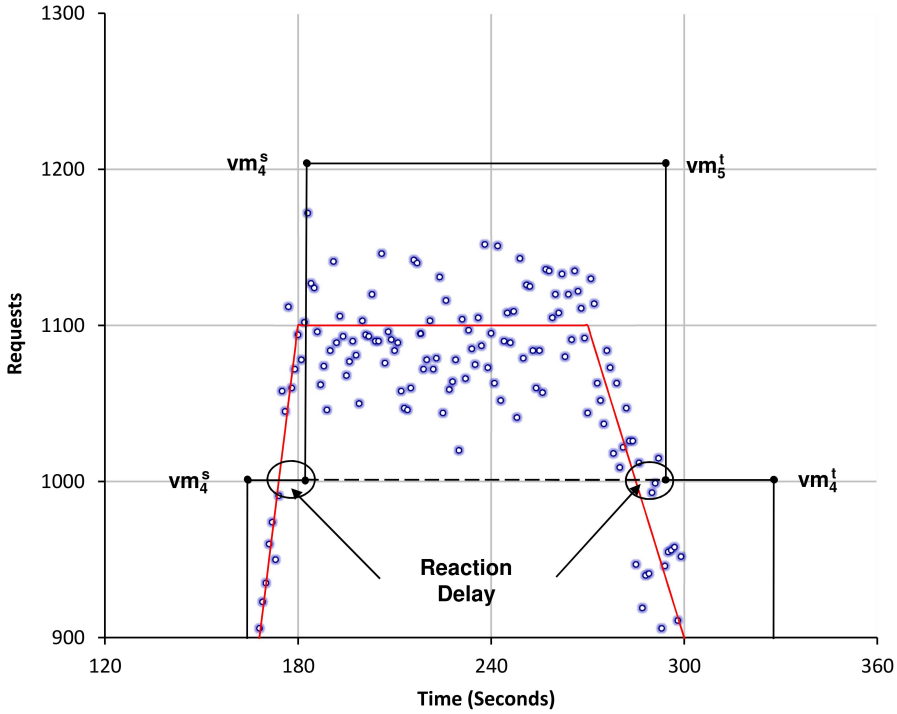


Figure 7.10. Properties of on Demand Scaling

More specifically the overhead associated with starting and stopping VMs will have a direct impact on the ability of the system to adhere to KPI thresholds.

Figure 7.11 illustrates a theoretical Cloud usage trace using the above defined model with varying changes in demand over time that could be used to test the proposed enhanced Pathology architecture realistically. A flash crowd event increases load on the system over the period of a few days that is beyond the normal expect load. The normal expect load on the system oscillates with an assumed peak demand on the system during the hours of 5pm to 11pm and requests are again generated using a Poisson distribution.

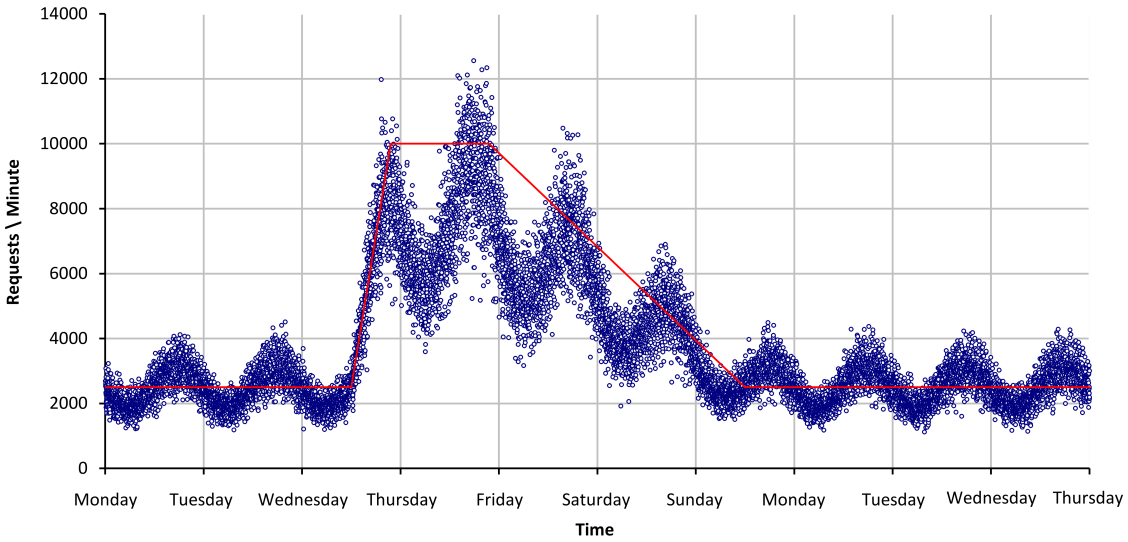


Figure 7.11. Theoretical Usage Trace

As demand on the system increases or deceases, the application layer will induce differing loads on the data layer. Altering the thresholds of the data layer KPIs, used to decide when to start and stop resources, will provide insight into this relationship and expose how the performance of the application layer will be effected.

7.4 Evaluation

In the following section of the thesis, the aforementioned Globus Virtual Cluster is evaluated against Nimbus and the performance of the Pathology application is disclosed as a baseline for the previously discussed enhanced architecture.

7.4.1 Globus Virtual Cluster

To assess the validity of the Globus Virtual Cluster (GVC) solution, this subsection provides a feature comparison with Nimbus enabled Grid clusters and discusses the limited IaaS provider interoperability of Nimbus when compared to GVC.

7.4.1.1 Feature Comparison Against Nimbus Clusters

Nimbus is an open-source toolkit for providing IaaS capabilities tailored to the scientific community. It focuses on three goals:

i. Enable academic resource providers to build private and community IaaS Clouds.

Two complementary tools are available for use: the Nimbus Workspace Service provides access to compute resources and Cumulus a quota based storage system.

ii. Enable academic users to access both private and public Cloud resources. This is achieved via the Context Broker [179] tool, which provides common configuration and security context across Cloud resources providers.

iii. Enable developers to extend and customise Nimbus via a highly configurable and extendible open source IaaS implementation. Resources are manageable via traditional resource schedulers such as PBS and via an interface that mimics the *de facto* standard of Amazon EC2.

Nimbus provides a number of features relevant to and aid in the deployment of Grids resources on to Clouds that are comparable to those of the GVC solution:

- **One-click Clusters**: Allows a Cloud client to launch "one-click" clusters of VMs. Each node is securely configured to operate in new network and security environment.
- **Configuration Management**: Optional mechanisms to perform configuration actions such as DHCP network assignment and arbitrary image customization where by files can be inserted in an image.
- **VM Network Configuration**: A collection of tools to assign network attributes to a VM at deployment time.

The Context Broker [179] service is primarily used to configure and coordinate the lunching of virtual cluster resources and provide these features. The broker stores a number of cluster configuration files that can be altered by a user to define the cluster size and networking attributes. Contextualization is achieved via a Context Agent [178], a boot time script that runs on each VM. The agent securely contacts the Context Broker service using a secret key. This key is created on the fly and seeded into each VM instance. The agent gathers context data on the cluster configuration from the Context Broker and applies this as boot-time changes to a VM.

Comparing these features, the Globus Virtual Cluster is preconfigured and automatically contextualized without relying on a pervasive external runtime contextualization mechanism that requires modification of a provider's IaaS software stack to support it. Instead VM network configuration is applied via encoding a NIC's MAC address in the case of a static environment, as discussed in Section 6.3, or applied automatically by detecting the presence of a DHCP server. GVC does not require the user to understand

how to configure the Globus Toolkit and its software supporting stack, it is usable "*out-of-the-box*". In addition clusters within Nimbus require network connectivity between the Context Agent and the Context Broker to operate, GVC does not. The GVC is pre-contextualised before deployment time and can be used in a virtualised desktop environment for development and testing purposes, where network connectivity maybe limited.

7.4.1.2 Provider Interoperability

There is a need for IaaS provider interoperability when deploying Grid applications on to a Cloud so that more advanced scenarios such as Federated and Multi-Clouds (see Section 6.3.3.1 for details), with the benefits of resource sharing and increased scalability, can be supported.

There are limitations inherent within Nimbus that prevent its clustering mechanism from being interoperable with other Cloud infrastructures. Nimbus does feature interoperability with other IaaS via a Web Service interfaces that mimic Amazon EC2 but does not provide interoperable contextualization. Nimbus instead provides a generic contextualization mechanism that can only be used with the Nimbus software stack in private and hybrid Cloud scenarios where end to end network connectivity is available.

The GVC enables the use of Grid Middleware on Cloud resources across organisational boundaries and maintains interoperability by being agnostic of the underlying IaaS and Hypervisor software.

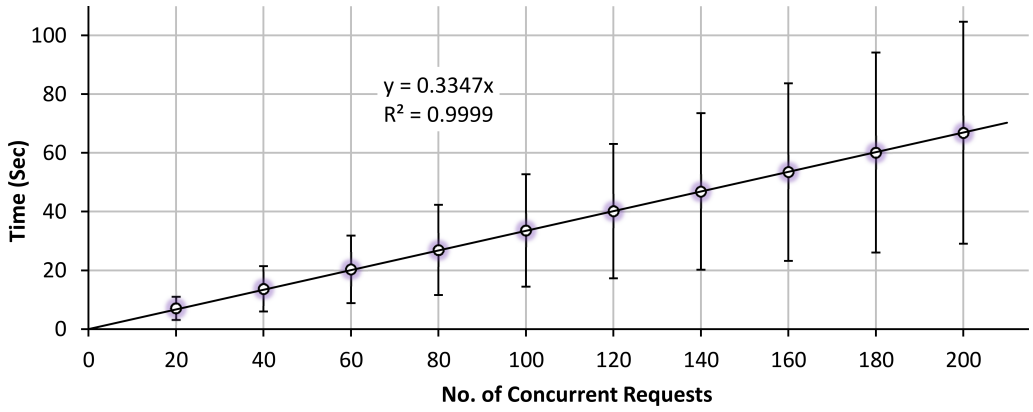
7.4.2 Pathology Application Performance

The objective of the following subsection is to ascertain the performance of the Pathology application in Section 7.3 for later comparison with the Cloud enhanced version. The experiments were performed on a Quad Core Intel Q9550 at 2.83Ghz with 4GB of DDR3 1333Mhz memory and a Seagate Barracuda 7200rpm 500GB hard disk drive. The experimental design is as follows:

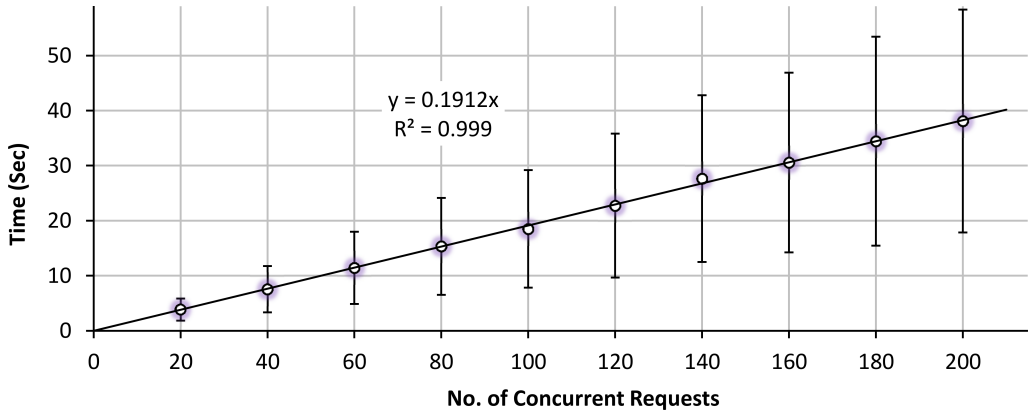
- Requests are made for a single 2000 by 2000 pixel sector from 1, 2 and 4 gigapixel (filesize: 1–1.1GByte) images concurrently.
- Requests are made for 1, 2, 4 different 2000 by 2000 pixel sector concurrently from a single mega pixel image.
- The number of concurrent requests are varied from 20 to 200 in increments of 20 spread evenly across each mega pixel images or 2000 by 2000 pixel sector.

- Ten iterations of the experiment are performed.

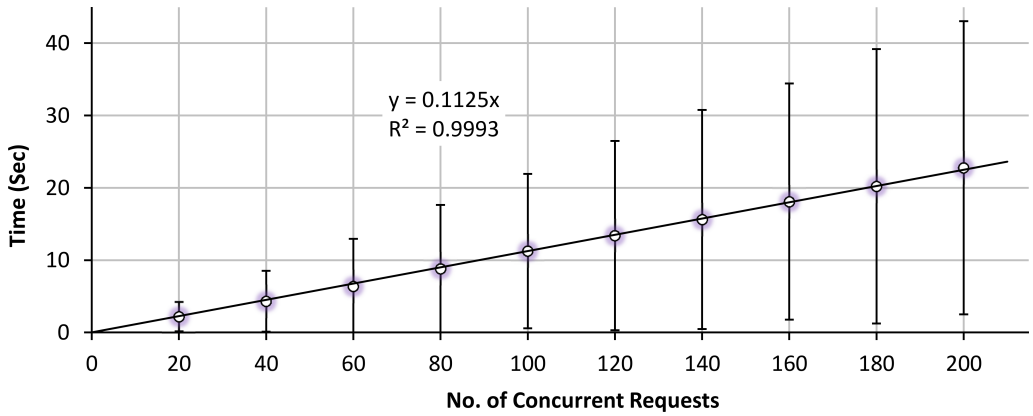
Figure 7.12 illustrate the average times to service a concurrent number of requests for a sector of a gigapixel image. The results provide insight into the performance and design of Aperio Image Server.



(a)



(b)



(c)

Figure 7.12. Time to service requests across a) 1 Image, b) 2 Images and c) 4 Images.

It can be seen when comparing Figures 7.12(a), 7.12(b) and 7.12(c) that increasing the number of images accessed concurrently from one to two to four respectively induces a linear reduction in the average request time. Figure 7.12(b) shows a 43% reduction in the average time to service a request split between 2 images compared with that of a single image in Figure 7.12(a). Likewise, Figure 7.12(c) shows a 41% reduction in the average time to service a request split between four images compared to Figure 7.12(b). Further comparing Figures 7.12(a) and 7.13, where the number of concurrently accessed sectors increase from one to two to four; no change in the average service time of a request is seen. Thus, it can be deduced from the results: increasing the number of concurrently accessed images reduces the average time to service a request, if the requests are spread evenly between the images and that increasing the number of concurrently accessed sectors in an image yields no change in the average request time.

Figure 7.12 features a linear increase in the average request time. Each request showed an increase in the time to service from the last, illustrated further in Figure 7.13. This behaviour suggests file level locking of the images and would account for the serialisation of requests on a per image basis.

The error bars present in Figure 7.12 show the variance in the time to service a request through all ten iterations of the experiment. It can be seen that as the number of concurrent requests increases so does the variance in the time to service a request. A possible explanation for this could be socket contention between concurrent HTTP requests at the TCP level, where by additional latency is incurred due to the overheads induced by polling the state of a large number of sockets.

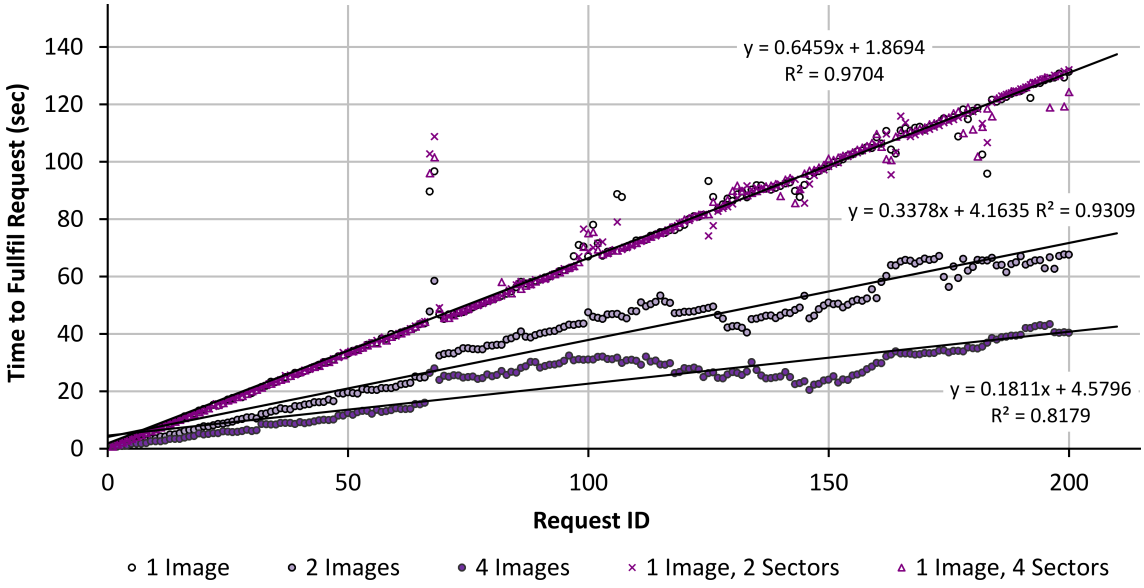


Figure 7.13. Individual Service Times for a Request as Part of 200 Concurrent Requests.

Using these results, it can be seen that by increasing the number of Aperio Image Server binaries, as described in the Cloud enhanced version of the pathology application, could alleviate the poor scalability exhibited. This would avoid image level locking if a number of pathologists were to access the same slide concurrently and reduce the overall response time. Further improvements could be made to the response time of a request by limiting each Aperio server instances to 60 concurrent requests where variance is at an acceptable level, as illustrated in Figure 7.13.

7.5 Summary

In this chapter the motivation behind and benefits of using Cloud Computing for academic research was discussed. The concept of a Virtual Grid, where Grid Middleware runs on Cloud infrastructure, was introduced. The standards and components of the Globus Toolkit usable in the creation of Grid infrastructure were highlighted. More over, a discussion of the evolution of the toolkit was presented. An implementation of a Virtual Grid using the Globus Toolkit and its standards and components, the Globus Virtual Cluster, was presented and then later evaluated against Nimbus. Nimbus although capable of creating Virtual Grids, is limited by its design with regards to Cloud provider interoperability and ease of use by academic users.

Furthermore, this Chapter has highlighted the relevance of the configurable parameters discussed in Chapters 3 and 4. Issues of Grid resource collections, their connectivity to one another and the continuous monitoring of these resources were all considered in Section 7.2. In addition, issues of resource discoverability and the selection of API's at runtime were acknowledge in Section 7.3.

In addition, the issues surrounding academic legacy applications in the Cloud were discussed and highlighted via a Pathology application that exhibits poor scalability and performance. A Cloud enhanced architecture enabled via contextualisation was proposed to overcome the scalability and performance issues. A model of demand for the purpose of testing the scalability of the proposed architecture was developed and performance results of the native pathology application were presented for future comparison with the implemented Cloud enhanced architecture.

Chapter 8

Conclusion

8.1 Summary

The work presented in this thesis demonstrates the performance and scalability of two open source IaaS implementations and two associated Hypervisors, including the overheads associated with their Virtualization technologies relating to block device I/O Paravirtualization. In addition, contextualization was presented as a mechanism to improve the QoS obtainable from an application by improving its (re-)configurability, enabling more complex applications deployments that exhibit improved scalability, availability and performance. In addition, contextualization enabled the deployment of Grid Middleware onto Cloud resources, while maintaining provider interoperability. The highlights of the thesis are as follows:

- **Chapter 2**: Introduced the concept of Cloud Computing, its classifications, deployment models and a range of Cloud types. The landscape of Cloud architectures both open source and commercial were presented covering all levels of the Cloud Service Stack, including IaaS, PaaS and SaaS. A number of End-User applications were presented that make use of Cloud Computing. Finally, the topic of Resource Management within Cloud Computing including the importance elasticity plays in scalability, resource scheduling in an open source and commercial Cloud schedulers and some tools for Cloud resource monitoring, were discussed. Additionally, the chapter presented the concept of Virtualization and virtual resource manage-

ment in the context of Cloud Computing. The technology behind Virtualization, the Hypervisor, was discussed and classified as either *Type 1* or *Type 2*, along with the techniques used to virtualise resources. The importance of migration and live migration of VMs was presented. Finally, a survey of contemporary Hypervisors, both open source and commercial, was presented including a discussion on their applicability to Cloud Computing.

- **Chapter 3**: Introduced Cloud application composition and the heritage it draws on from within the topic of Service Oriented Architectures. Service Orientation and Web Service technology was discussed in detail, including the technologies that enable their use. Cloud Engineering was discussed as a systematic approach to the creation and composition of Cloud applications and a number of issues to consider when contemplating the use of Cloud Computing were presented. In addition, a number of Cloud simulators were discussed as tools to ascertain the performance of an application before development and/or deployment.

- **Chapter 4**: Discussed the paradigm of Grid Computing, including the architectural philosophies and applications that make use of it. In addition, Resource Management in Grids including the topics of QoS, resource allocation and monitoring were discussed. The relevance of Service Level Agreements in Grids was related back to Cloud Computing. More so, a survey of Grid Middleware and the role that it plays within Grid Computing was presented. Finally, a comparison of Grids and Clouds was discussed, highlighting a need for research into how these paradigms can complement each other.

- **Chapter 5**: Discussed the performance of Cloud infrastructure, identifying a number of issues and additionally, related work on Cloud performance was critiqued. The topic of Virtual Machine image management was introduced, along side the storage systems used to store the images within an IaaS provider. The performance overheads associated with the propagation of images, from an image repository to a host machine, in two Cloud infrastructure management systems, was discussed and compared. In addition, a number of image maintenance and management issues were discussed. The topic of high performance Virtualization within Cloud infrastructure was presented and two contemporary Hypervisors were compared. In addition the topic of Block I/O Paravirtualization was discussed, as a performance limiting factor in Cloud Computing virtual resources. Finally, performance evaluations of Virtual Machine image formats, the resource provisioning and propagation delays in two Cloud management technologies in addition to the overheads of vir-

tual block I/O devices, were presented. These evaluations highlighted a number of areas for performance and scalability improvement in Cloud Computing.

- **Chapter 6**: Introduced the concept of contextualization as a mechanisms to enable the autonomous configuration of Cloud Computing software and discusses the landscape of tools available to help manage the complexity and scale of Cloud Computing systems. An generalised architecture for the contextualization of a range of Cloud software was presented. An implementation based on this architecture, the Contextualization Tools, and the non-functional requirements it enables the fulfilment of, was discussed. Additionally, the concept of Recontextualization, a mechanism and an architecture was presented, as a way to enable the reconfiguration of a system after deployment. A number of Recontextualization approaches were discussed and one showing the most promise was selected and implemented. Finally, the performance of the Contextualization and Recontextualization implementations were evaluated and the functionality they provide compared to related Cloud technologies.

- **Chapter 7**: Discussed the application of Cloud Computing resources in research through the execution of Grid Middleware on the Cloud. The concept of a Virtual Grid was introduced and an implementation, the Globus Virtual Cluster, presented. Additionally, the implications of running legacy applications were discussed, in light of a model for generating Cloud usage traces. A legacy application, used by Pathologists for the purpose of distributing parts of gigapixel images of tissue samples for analysis, was discussed including the benefits the application can leverage from using Cloud Computing technologies. In addition a Cloud architecture for enhancing the pathology application's scalability and performance was presented. Finally, an evaluation of the Globus Virtual Cluster with an emphasis on comparing its features with existing solutions and its interoperability, in addition to the performance of the legacy pathology application, were presented.

8.2 Research Contributions

The objectives of this thesis, as discussed in Section 1.2, were to discover issues relating to QoS and Resource Management in Cloud Computing, then improve the QoS provisioned by a Cloud Provider and received by a Cloud End-User, beyond current best-effort practices. During the research, it became apparent that a number of contributions were needed to support these objectives. The contributions are summarised as following:

- Experimental results on the performance and scalability of the two open source IaaS solutions: OpenNebula and Nimbus, with regards to the delays associated with provisioning VM resources, for the purpose of ascertaining the suitability of these technologies to provide QoS to an End-User.
- A performance evaluation of the Paravirtualization techniques used in the two Hypervisor: Xen and KVM, for the previsioning of performant block devices to VMs. An area of research that looks the most promising for providing the largest improvement to QoS in Cloud Computing.
- A performance evaluation on the range of VM image formats available for use in a Cloud application, for the purpose of ascertaining their suitability for supporting the performance based QoS requirements of an End-User's application.
- A Contextualization tool software prototype that enables the configuring of a VM's hardware and embedded application, including any PaaS dependencies, at Deployment time.
- A prototype Recontextualization mechanism for reconfiguring an application and its supporting software dependencies after live migration during the Operation phase of the service life-cycle.
- A Virtual Grid prototype, the Globus Virtual Cluster, which enables the deployment of Grid applications, for academic use cases, onto public and private Cloud providers. The prototype maintains interoperability from the provider's resource management stack as it has no dependencies on the underlying IaaS technology.
- A performance evaluation of a Cloud enhanced architecture for a legacy scientific application used by Pathologists that currently exhibits poor End-User QoS with regards to scalability and fault tolerance.
- A model for the generation of realistic workloads when simulating the demand placed on a Cloud application and its supporting PaaS and IaaS stack.

8.3 Future Work

There are many ways in which the work presented in this thesis can be extended. The most promising extensions are listed bellow:

8.3.1 Additional Evaluations On Cloud Infrastructure

Extension could be made to the work on Hypervisor performance to evaluate the performance of the newly released XEN 4.2 that sees the introduction of new version of the redesigned block device backend, blktap2, which due to time constraints could not be included in this thesis. In addition, the trade off between the features, such as compression and encryption, of enhanced image formats, such as QCOW2, VMware's VMDK and Microsoft's VHD, could be assessed. This could in turn lead to additional work evaluating the performance of Cloud resource migration and fault tolerance techniques and how these are effected by the implementation of VM image formats. Additionally, further work could be carried out to evaluate the overheads surrounding other popular IaaS solutions such as Eucalyptus [71]. Further more, experiments on the ability of open source IaaS solutions to manage increase numbers of physical resource, could provide interesting insights into the scalability of these solutions, at the scales that true Cloud providers operate. This was not feasibly within this thesis due to lack of funds to access such a resource pool large enough to perform the experiments. Finally, an in-depth survey of source code, investigating and analysing the root causes of many of the performance related issues discovered in the evaluations, could shed light on what specific improvements are needed to be made to enhance QoS in future software releases.

8.3.2 Extended Contextualization

Future work could be made to the Contextualization Tools to extend the existing architecture to enable support for a greater number of image formats and Hypervisors. In addition, further integration with other core components of the OPTIMIS toolkit, such as the tools responsible for VM level monitoring, could provide addition support for more non-functional and functional application requirements. Finally, plans to implement advanced caching mechanisms that improve the ability of the tools to reuse existing contextualized images, could increase the general applicability of the solution to a wider number of applications that require rapid deployment.

8.3.3 Integrated Recontextualization

Future work could include creating a unified mechanism for Contextualization and Recontextualization, integrating the generalised solution with a number of major software projects, including the OPTIMIS toolkit. In addition, Recontextualization mechanisms for the dynamic binding of PaaS APIs could be explored. Finally, further studies and

improvements on the implemented approach could be evaluated to reduce the overhead imposed by Recontextualization.

8.3.4 Globus Virtual Cluster Overheads

Further work on the Globus Virtual Cluster could entail evaluating the overheads of running a number of Grid based applications or HPC benchmarks on the prototype. In addition, deploying the Globus Virtual Cluster on a number of public Cloud providers could provide further insight into its scaling behaviour, as the number of Cloud resources it uses grow above the eight machines available in the Leeds Cloud Testbed.

8.3.5 Cloud Enhanced Pathology Application

Future work on the Pathology application could involve the full implementation of the proposed Cloud enhanced architecture, in addition to carrying out a comparative performance evaluation. The performance evaluation could make use of the proposed model of user demand and the results of which could be compared to original implementation to ascertain any reductions in overheads or improvements in scalability in both computation and storage resource dimensions.

Bibliography

[1] A. Afzal, A.S. McGough, and J. Darlington. Capacity planning and scheduling in grid computing environments. *Future Generation Computer Systems*, 24(5):404 – 14, 2008.

[2] I. Ahmad, J.M. Anderson, A.M. Holler, R. Kambo, and V. Makhija. An analysis of disk performance in VMware ESX server virtual machines. In *2003 IEEE International Workshop on Workload Characterization*, pages 65 – 76, October 2003.

[3] Robert Aiello and Leslie Sachs. *Configuration Management Best Practices: Practical Methods that Work in the Real World*. Addison-Wesley Professional, 1st edition, 2010.

[4] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link. The Globus Striped GridFTP Framework and Server. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pages 54 – 54, November 2005.

[5] Amazon Web Services. `http://aws.amazon.com`, June 2012.

[6] High Performance Computing (HPC) on AWS. `http://aws.amazon.com/hpc-applications/`, August 2012.

[7] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the American Federation of Information Processing Societies joint computer conference (AFIPS '67)*, AFIPS '67 (Spring), pages 483 – 485, New York, NY, USA, 1967. ACM.

[8] AMI - Amazon Machine Image Format. `http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/creating-an-ami.html`, June 2012.

[9] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Kakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Spec-

ification (WS-Agreement). Technical report, Open Grid Forum, May 2007. http://www.ogf.org/documents/GFD.107.pdf.

[10] Apache Software Foundation. `http://www.apache.org/`, September 2012.

[11] Aperio - Digital Pathology Services for Healthcare and Life Sciences. `http://www.aperio.com/`, August 2012.

[12] Apple iCloud. `http://www.apple.com/icloud/`, September 2012.

[13] CA Applogic. `http://www.ca.com/gb/cloud-platform.aspx`, September 2012.

[14] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[15] D. Armstrong, K. Djemame, S. Nair, J. Tordsson, and W. Ziegler. Towards a Contextualization Solution for Cloud Platform Services. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 328–331. IEEE, 2011.

[16] Shah Asaduzzaman and Muthucumaru Maheswaran. Heuristics for Scheduling Virtual Machines for Improving QoS in Public Computing Utilities. In *9th International Conference on Computer and Information Technology ICCIT-2006*, December 2006.

[17] AssessGrid - Advanced Risk Assessment and Management for Trustable Grids. `http://www.assessgrid.eu/`, June 2012.

[18] Muhammad Atif and Peter Strazdins. Optimizing live migration of virtual machines in SMP clusters for HPC applications. In *NPC 2009 - 6th International Conference on Network and Parallel Computing*, pages 51 – 58, Gold Coast, QLD, Australia, 2009.

[19] F. Azmandian, M. Moffie, J.G. Dy, J.A. Aslam, and D.R. Kaeli. Workload Characterization at the Virtualization Layer. In *IEEE 19th International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS '11)*, pages 63 – 72, July 2011.

[20] Azure Services Platform. `http://www.microsoft.com/azure`, June 2012.

[21] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164 – 177, 2003.

[22] Borja Sotomayor Basilio. A Resource Management Model for VM-based Virtual Workspaces. Master's thesis, The University of Chicago, Department of Computer Science, February 2007.

[23] Steffen Bernius and Julia Krnung. Fostering Academic Research by Cloud Computing - The User's Perspective. In *European Conference on Information Systems, ECIS '12*, 2012.

[24] Nikhil Bhatia and Jeffrey S. Vetter. *Virtual Cluster Management with Xen*, volume 4854 of *Lecture Notes in Computer Science*, chapter 25, pages 185 – 194. Springer Berlin / Heidelberg, March 2008.

[25] Big Data Definition. `http://mike2.openmethodology.org/wiki/Big_Data_Definition`, September 2012.

[26] J. Bisbal, D. Lawless, Bing Wu, and J. Grimson. Legacy information systems: issues and directions. *Software, IEEE*, 16(5):103 – 111, Sep / Oct 1999.

[27] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehorster, and A. Brinkmann. Non-intrusive Virtualization Management using libvirt. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 574 – 579, Piscataway, NJ, USA, 2010.

[28] Bonnie++ - Benchmark Suite. `http://www.coker.com.au/bonnie++/`, June 2010.

[29] Greg Boss, Padma Malladi, Dennis Quan, Linda Legregni, and Harold Hall. Cloud Computing. White paper, IBM High Performance On Demand Solutions (HiPODS), October 2007.

[30] Kevin D. Bowers, Ari Juels, and Alina Oprea. HAIL: A high-availability and integrity layer for cloud storage. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 187 – 198, Chicago, IL, United states, 2009.

[31] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the 3rd international conference on Virtual execution environments*, pages 169–179. ACM, June 2007.

[32] James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Market-oriented grids and utility computing: The state-of-the-art and future directions. *Journal of Grid Computing*, 6(3):255 – 276, 2008.

[33] Michael L. Brodie and Michael Stonebraker. *Migrating legacy systems: gateways, interfaces & the incremental approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[34] Mark Burgess. Computer Immunology. In *Proceedings of the 12th USENIX conference on System administration*, LISA '98, pages 283 – 298, Berkeley, CA, USA, 1998. USENIX Association.

[35] Mark Burgess. Configurable Immunity For Evolving Human-Computer Systems. *Science of Computer Programming*, 51(3):197 – 213, June 2004.

[36] Bandwidth Monitor NG. `http://www.gropp.org/?id=projects&sub=bwm-ng`, June 2012.

[37] Claudio Cacciari, Francesco D'Andria, Bjšrn Hagemeier, Daniel Mallmann, Josep Martrat, David Garc'a PerŽz, Angela Rumpl, Wolfgang Ziegler, and Csilla Zsigri. Software licenses as mobile objects in distributed computing environments. In *Euro-Par 2010 workshops*. Springer, Berlin, 2010.

[38] Rodrigo N. Calheiros, Rajiv Ranjan, Cesar A. F. De Rose, and Rajkumar Buyya. CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. Technical report, The Grid Computing and Distributed Systems (GRIDS) Laboratory, University of Melbourne, March 2009.

[39] Cloud Computing Interoperability Forum (CCIF). `http://www.ccif.org/`, September 2012.

[40] CFEngine 3 - Configuration Management Software for Agile System Administrators. `http://cfengine.com/`, August 2012.

[41] Chef - A Systems Integration Framework. `http://wiki.opscode.com/display/chef/Home`, August 2012.

[42] Yang Chen, Tianyu Wo, and Jianxin Li. An Efficient Resource Management System for On-line Virtual Cluster Provision. In *IEEE 2009 International Conference on Cloud Computing*, 2009.

[43] P. Chomphuwiset, D. Treanor, and D.Magee. Context-Based Classification of Cell Nuclei and Tissue Area in Liver Histology. In *Medical Image Understanding and Analysis*, 2011.

[44] Hao-Hua Chu and K. Nahrstedt. CPU service classes for multimedia applications. In *Multimedia Computing and Systems, 1999. IEEE International Conference on*, volume 1, pages 296 – 301, Jul 1999.

[45] L. Chunlin and L. Layuan. Cross-layer optimization policy for QoS scheduling in computational grid. *Journal of Network and Computer Applications*, 31(3):258 – 84, August 2008.

[46] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, May 2005.

[47] S. Clayman, A. Galis, C. Chapman, G. Toffetti, L. Rodero Merino, L.M. Vaquero, K. Nagin, and B. Rochwerger. Monitoring Service Clouds in the Future Internet. In *Towards the Future Internet - Emerging Trends from European Research*, pages 115–126, Amsterdam, The Netherlands, 2010. IOS Press.

[48] Cloud Security Alliance. `https://cloudsecurityalliance.org/`, September 2012.

[49] CloudStack - Open Source Cloud Computing with Apache CloudStack. `http://cloudstack.org/`, September 2012.

[50] Murray Cole. *Algorithmic skeletons: structured management of parallel computation*. MIT Press, Cambridge, MA, USA, 1991.

[51] David Colling, T Ferrari, Youssef Hassoun, Chenxi Huang, C Kotsokalis, Stephen McGough, Yash Patel, E Ronchieri, and P Tsanakas. On Quality of Service Support for Grid Computing. In *2nd International Workshop on Distributed Cooperative Laboratories and Instrumenting the GRID (INGRID 2007)*, April 2007.

[52] Contrail - Open Computing Infrastructures For Elastic Services. `http://contrail-project.eu/`, June 2012.

[53] J.W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications, 2002.

[54] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. Remus: high availability via asynchronous virtual machine replication. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161 – 174, Berkeley, CA, USA, 2008. USENIX Association.

[55] S. De Gyves Avila and K. Djemame. A QoS Optimization Model for Service Composition. In *4th International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE'2012)*, Nice, France, July 2012.

[56] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing On Large Clusters. *Commun. ACM*, 51(1):107 – 113, January 2008.

[57] Kemal A. Delic and Martin Anthony Walker. Emergence of the Academic Computing Clouds. *Ubiquity*, 9(31):1 – 4, August 2008.

[58] T. Deshane, Z. Shepherd, J. Matthews, M. Ben Yehuda, A. Shah, and B. Rao. Quantitative comparison of Xen and KVM. In *Xen Summit*, Berkeley, CA, USA, June 2008. USENIX Association.

[59] Giuseppe Di Modica, Orazio Tomarchio, and Lorenzo Vita. Dynamic SLAs management in service oriented environments. *Journal of Systems and Software*, 82(5):759 – 771, 2009.

[60] Karim Djemame, Iain Gourlay, James Padgett, Georg Birkenheuer, Matthias Hovestadt, Odej Kao, and Kerstin Voss. Introducing Risk Management into the Grid. In *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, page 28, Washington, DC, USA, 2006. IEEE Computer Society.

[61] Distributed Management Task Force (DMTF). `http://www.dmtf.org/`, September 2012.

[62] Yaozu Dong, Jinquan Dai, Zhiteng Huang, Haibing Guan, Kevin Tian, and Yunhong Jiang. Towards high-quality I/O virtualization. In *ACM International Conference Proceeding Series*, pages 12 – 22, Haifa, Israel, 2009.

[63] Dong-Jae Kang, Chei-Yol Kim, Kang-Ho Kim, and Sung-In Jung. Proportional disk I/O bandwidth management for server virtualization environment. In *2008 International Conference on Computer Science and Information Technology*, pages 647 – 53, Piscataway, NJ, USA, 2008.

[64] Dropbox - Cloud Storage. `https://www.dropbox.com/`, September 2012.

[65] N. Dube and M. Parizeau. Utility computing and market-based scheduling: shortcomings for grid resources sharing and the next steps. In *2008 22nd International Symposium on High Performance Computing Systems and Applications (HPCS '08)*, pages 59 – 68, Piscataway, NJ, USA, 2008.

[66] Enabling Grids for E-SciencE (EGEE). `http://www.eu-egee.org/`, September 2012.

[67] I.P. Egwutuoha, Shiping Chen, D. Levy, and B. Selic. A Fault Tolerance Framework for High Performance Computing in Cloud. In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*, pages 709 – 10, Los Alamitos, CA, USA, 2012.

[68] Erik Elmroth and Lars Larsson. Interfaces for placement, migration, and monitoring of virtual machines in federated clouds. In *8th International Conference on Grid and Cooperative Computing, GCC 2009*, pages 253 – 260, Lanzhou, Gansu, China, 2009.

[69] Enomaly - Elastic Computing Platform. `http://www.enomaly.com`, June 2012.

[70] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[71] Eucalyptus - Elastic Utility Computing Architecture. `http://www.eucalyptus.com/`, June 2012.

[72] Cloud Computing Storage And The Exponential Growth Of Data. `http://www.cloudtweaks.com/2010/09/cloud-computing-storage-and-the-exponential-growth-of-data/`, September 2012.

[73] M. Fenn, S. Goasguen, and J Lauret. Contextualization in Practice: The Clemson Experience. In *Proceedings of the 13th International Workshop on Advanced*

Computing and Analysis Techniques in Physics Research (ACAT '10), Jaipur, India, 2010.

[74] A.J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali Eldin, C. Zsigri, R. Sirvent, J. Guitart, R.M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S.K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forgó, T. Sharif, and C. Sheridan. OPTIMIS: a Holistic Approach to Cloud Service Provisioning. *Future Generation Computer Systems*, 28:66 – 77, 2011.

[75] Flexiant - Your Cloud Simplified. `http://www.flexiant.com/`, September 2012.

[76] Flexiscale - Utlity Computing On Demand. `http://www.flexiscale.com`, September 2012.

[77] I. Foster. *The Physiology of the Grid, in Grid Computing: Making the Global Infrastructure a Reality*. J. Wiley, New York, 2003.

[78] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *FIP International Conference on Network and Parallel Computing*, 2005.

[79] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *Lecture Notes in Computer Science*, 3779:2–13, 2005.

[80] I. Foster, Yong Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop*, page 10, Piscataway, NJ, USA, 2008.

[81] Ian Foster. What is the Grid? A Three Point Checklist. GridToday, June 2002.

[82] Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200 – 222, 2001.

[83] Frank Shu and Nathan Obr. Data Set Management Commands Proposal For ATA8-ACS2. Specification Revision 6, INCITS - InterNational Committee for Information Technology Standards, December 2007. `http://t13.org/Documents/UploadedDocuments/docs2008/e07154r6-Data_Set_Management_Proposal_for_ATA-ACS2.doc`.

[84] Ganglia Monitoring System. `http://www.nagios.org/about/overview`, September 2012.

[85] Gartner. Gartner Survey Shows Data Growth as the Largest Data Center Infrastructure Challenge. `http://www.gartner.com/it/page.jsp?id=1460213`, September 2012.

[86] Jeremy Geelan. Twenty-One Experts Define Cloud Computing. Electronic Journal, June 2010.

[87] W. Gentzsch. Sun Grid Engine: Towards creating a compute power grid. In *First IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001. Proceedings.*, pages 35 –36, 2001.

[88] The Globus Toolkit - An open source software toolkit for building grids. `http://www.globus.org/toolkit/`, June 2012.

[89] Globus Alliance. `http://www.globus.org`, June 2010.

[90] Crux for GT Developers. `http://confluence.globus.org/display/whi/Crux+for+GT+Developers`, August 2012.

[91] GlusterFS - An Open Source, Scalable Distributed File System. `http://www.gluster.org/`, August 2012.

[92] I. Goiri, F. Julia, J. Ejarque, M. de Palol, R.M. Badia, J. Guitart, and J. Torres. Introducing virtual execution environments for application lifecycle management and SLA-driven resource distribution within service providers. In *Proceedings 2009 Eighth IEEE International Symposium on Network Computing and Applications (NCA)*, pages 211 – 18, Piscataway, NJ, USA, 2009.

[93] R.P. Goldberg. *Architectural Principles for Virtual Computer Systems*. PhD thesis, Harvard University, Cambridge, MA, 1972.

[94] Z Gong, X. Gu, and J. Wilkes. PRESS: PRedictive Elastic ReSource Scaling for cloud systems. In *Proceedings of the 6th International Conference on Network and Service Management (CNSM '10)*, pages 9 – 16, Piscataway, NJ, USA, 2010.

[95] Google Apps. `http://www.google.com/apps/business`, June 2012.

[96] Google Apps Engine. `http://code.google.com/appengine`, June 2012.

[97] Google Docs. http://docs.google.com, September 2012.

[98] Google Drive. https://drive.google.com, September 2012.

[99] GRAM - Grid Resource Allocation and Management Protocol. http://dev.globus.org/wiki/GRAM, August 2012.

[100] GT Information Services: Monitoring & Discovery System (MDS). http://www.globus.org/toolkit/mds/, August 2012.

[101] GRAM - Version 2. hhttp://www.globus.org/toolkit/docs/2.4/gram/, August 2012.

[102] GRAM - Version 4. http://www.globus.org/toolkit/docs/4.2/4.2.1/execution/gram4/, August 2012.

[103] GRAM - Version 5. http://www.globus.org/toolkit/docs/5.0/5.0.0/execution/gram5/, August 2012.

[104] T. Grance and P. M. Mell. The NIST Definition of Cloud Computing. Technical report, National Insitute of Standards and Technology (NIST), September 2011.

[105] GRIA - Service Oriented Collaborations for Industry and Commerce. http://www.gria.org/, June 2012.

[106] GridSim - Grid Simulation Toolkit. http://www.gridbus.org/gridsim, June 2012.

[107] GridWay. http://www.gridway.org/, September 2012.

[108] The Globus Toolkit - Version 2. http://www.globus.org/toolkit/docs/2.4/, August 2012.

[109] The Globus Toolkit - Version 3. http://www.globus.org/toolkit/docs/3.0/, August 2012.

[110] The Globus Toolkit - Version 4. http://www.globus.org/toolkit/docs/4.0/, August 2012.

[111] The Globus Toolkit - Version 5. http://www.globus.org/toolkit/docs/5.2/, August 2012.

[112] Ajay Gulati, Ganesha Shanmuganathan, Anne Holler, and Irfan Ahmad. Cloud-scale resource management: challenges and techniques. In *Proceedings of the 3rd USENIX conference on hot topics in cloud computing*, HotCloud'11, Berkeley, CA, USA, 2011. USENIX Association.

[113] Hadoop - Distributed File System. `http://hadoop.apache.org/core`, June 2012.

[114] Haizea - an open-source virtual machine-based lease management architecture. `http://haizea.cs.uchicago.edu/index.html`, June 2012.

[115] Jacob Gorm Hansen and Eric Jul. Lithium: virtual machine storage for the cloud. In *SoCC '10: Proceedings of the 1st ACM symposium on Cloud computing*, pages 15 – 26, New York, NY, USA, 2010. ACM.

[116] Eric Harney, Sebastien Goasguen, Jim Martin, Mike Murphy, and Mike Westall. The efficacy of live virtual machine migrations over the Internet. In *VTDC'07: Proceedings of the 3rd International Workshop on Virtualization Technology in Distributed Computing*, Reno, NV, United states, 2007.

[117] Mohammad Mehedi Hassan, Biao Song, Changwoo Yoon, Hyun Woo Lee, and Eui-Nam Huh. A novel market oriented dynamic collaborative cloud service infrastructure. In *SERVICES 2009 - 5th 2009 World Congress on Services*, number PART 2, pages 9 – 16, Bangalore, India, 2009.

[118] Parisa Heidari, Mathieu Desnoyers, and Michel Dagenais. Performance analysis of virtual machines through tracing. In *Canadian Conference on Electrical and Computer Engineering*, pages 261 – 266, Niagara Falls, ON, Canada, 2008.

[119] Michael R. Hines and Kartik Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 51 – 60, New York, NY, USA, 2009. ACM.

[120] Fred Howell and Ross Mcnab. simjava: A Discrete Event Simulation Library For Java. In *International Conference on Web-Based Modeling and Simulation*, pages 51 – 56, 1998.

[121] Wei Huang, Qi Gao, Jiuxing Liu, and Dhabaleswar K. Panda. High performance virtual machine migration with RDMA over modern interconnects. In *Proceedings*

- *IEEE International Conference on Cluster Computing, ICCC*, pages 11 – 20, Austin, TX, United states, 2007.

[122] Microsoft Hyper-V Server 2012. http://www.microsoft.com/en-us/server-cloud/hyper-v-server/, September 2012.

[123] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. www.globus.org/alliance/publications/papers/ogsa.pdf, Globus Alliance, 2002.

[124] IBM - Cloud Computing. http://www.ibm.com/ibm/cloud/ibm_cloud, June 2012.

[125] IBM - Computing on Demand. http://www-03.ibm.com/systems/deepcomputing/cod, June 2012.

[126] ICanCloud Simulator. http://icancloudsim.org, September 2012.

[127] T. Imada, M. Sato, and H. Kimura. Power and QoS performance characteristics of virtualized servers. In *Proceedings of the 2009 10th IEEE/ACM International Conference on Grid Computing (GRID)*, pages 232 – 40, Piscataway, NJ, USA, 2009.

[128] Aperio ImageServer Programmer's Reference. http://www.aperio.com/documents/api/Aperio_ImageServer_Programmer_Reference.pdf, August 2012.

[129] Emir Imamagic and Dobrisa Dobrenic. Grid infrastructure monitoring system based on Nagios. In *Proceedings of the 2007 Workshop on Grid Monitoring, GMW'07*, pages 23 – 28, Monterey, CA, United states, 2007.

[130] Intel. Best Practices for Paravirtualization Enhancements from Intel Virtualization Technology: EPT and VT-d. http://software.intel.com/en-us/articles/best-practices-for-paravirtualization-enhancements-from-intel-virtualization-technology-ept-and-vt-d/, June 2012.

[131] IOzone - Filesystem Benchmark. http://www.iozone.org/, June 2012.

[132] K.R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H.J. Wasserman, and N.J. Wright. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom '10)*, pages 159 – 168, December 2010.

[133] S. Jarvis, N. Thomas, and A. van Moorsel. Open issues in grid performability. *International Journal of Simulation: Systems, Science and Technology*, 5(5):3 – 12, December 2004.

[134] JavaSim & C++SIM. `http://javasim.codehaus.org/`, September 2012.

[135] Jianhua Che, Qinming He, Qinghua Gao, and Dawei Huang. Performance measuring and comparing of virtual machine monitors. In *2008 IEEE/IFIP 5th International Conference on Embedded and Ubiquitous Computing. EUC 2008*, volume 2, pages 381 – 6, Piscataway, NJ, USA, 2008.

[136] Hai Jin, Deng Li, Song Wu, Xuanhua Shi, and Xiaodong Pan. Live virtual machine migration with adaptive memory compression. In *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, New Orleans, LA, United states, 2009.

[137] Jin Cao, William S. Cleveland, Dong Lin, and Don X. Sun. Internet Traffic Tends To Poisson and Independent as the Load Increases. Technical report, Department of Statistics, Purdue University, 2001.

[138] Jun Zhu, Wei Dong, Zhefu Jiang, Xiaogang Shi, Zhen Xiao, and Xiaoming Li. Improving the Performance of Hypervisor-Based Fault Tolerance. In *Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 10 – 20, Piscataway, NJ, USA, 2010.

[139] P.A. Karger and D.R. Safford. I/O for virtual machine monitors: security and performance issues. *IEEE Security And Privacy*, 6(5):16 – 23, Sept.-Oct. 2008.

[140] Kate Keahey, Tim Freeman, Jerome Lauret, and Doug Olson. Virtual workspaces for scientific applications. In *SciDAC 2007 Conference*, Boston, MA, June 2007.

[141] G. Katsaros, G. Gallizo, R. Kübert, T Wang, J Oriol Fito, and D. Henriksson. A Multi-level Architecture for Collecting and Managing Monitoring Information in Cloud Environments. In *CLOSER 2011: International Conference on Cloud*

Computing and Services Science (CLOSER), Noordwijkerhout, The Netherlands, May 2011.

[142] K. Keahey and T. Freeman. Contextualization: Providing One-Click Virtual Clusters. In *Proceedings of the 4th IEEE International Conference on eScience (ESCIENCE '08)*, pages 301 – 308, Washington, DC, USA, 2008.

[143] A. Keller, K. Voss, D. Battre, M. Hovestadt, and O. Kao. Quality assurance of grid service provisioning by risk aware managing of resource failures. In *2008 Third International Conference on Risks and Security of Internet and Systems (CRiSIS 2008)*, pages 149 – 57, Piscataway, NJ, USA, 2008.

[144] J.O. Kephart and D.M. Chess. The Vision Of Autonomic Computing. *Computer*, 36(1):41 – 50, 2003.

[145] Mukil Kesavan, Ada Gavrilovska, and Karsten Schwan. Differential virtual time (DVT): rethinking I/O service differentiation for virtual machines. In *SoCC '10: Proceedings of the 1st ACM symposium on Cloud computing*, pages 27 – 38, New York, NY, USA, 2010. ACM.

[146] Carl Kesselman and Ian Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998.

[147] Ali Khajeh Hosseini, David Greenwood, James W. Smith, and Ian Sommerville. The Cloud Adoption Toolkit: supporting cloud adoption decisions in the enterprise. *Software: Practice and Experience*, 42(4):447 – 465, April 2012.

[148] Ali Khajeh Hosseini, Ian Sommerville, and Ilango Sriram. Research Challenges for Enterprise Cloud Computing. `http://arxiv.org/abs/1001.3257`, University of St Andrews, UK, 2010.

[149] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: The Linux Virtual Machine Monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.

[150] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135 – 164, February 2002.

[151] Donald Kossmann, Tim Kraska, and Simon Loesing. An evaluation of alternative architectures for transaction processing in the cloud. In *SIGMOD '10: Proceedings*

of the 2010 international conference on Management of data, pages 579 – 590, New York, NY, USA, 2010. ACM.

[152] S. Krishnan and J.C. Counio. Pepper: An Elastic Web Server Farm for Cloud Based on Hadoop. In *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom '10)*, pages 741 – 747, Los Alamitos, CA, USA, 2010.

[153] KVM - Kernel Based Virtual Machine. `http://www.linux-kvm.org`, June 2012.

[154] et al. Laure, E. Programming the Grid Using gLite. *Computational Methods in Science and Technology*, 12(1):33 – 45, 2006.

[155] Qiang Li and Yike Guo. Optimization of Resource Scheduling in Cloud Computing. In *Proceedings of the 2010 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC '10, pages 315 – 320, Washington, DC, USA, 2010. IEEE Computer Society.

[156] Qin Li, Jinpeng Huai, Jianxin Li, Tianyu Wo, and Minxiong Wen. HyperMIP: Hypervisor Controlled Mobile IP for Virtual Machine Live Migration across Networks. In *HASE '08: Proceedings of the 2008 11th IEEE High Assurance Systems Engineering Symposium*, 2008.

[157] Libvirt: The virtualization API. `http://libvirt.org/`, June 2012.

[158] Huan Liu and D. Orban. GridBatch: Cloud Computing for large-scale data-intensive batch applications. In *2008 8th International Symposium on Cluster Computing and the Grid (CCGRID '08)*, pages 295 – 305, Piscataway, NJ, USA, 2008.

[159] P. Liu, Z. Yang, X. Song, Y. Zhou, H. Chen, and B. Zang. Heterogeneous live migration of virtual machines. In *International Workshop on Virtualization Technology (IWVT'08)*, 2008.

[160] Ignacio M. Llorente, Ruben S. Montero, Eduardo Huedo, and Katia Leal. A grid infrastructure for utility computing. In *Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE*, pages 163 – 168, Manchester, United kingdom, 2006.

[161] Marek Wieczorek, Andreas Hoheisel, and Radu Prodan. Towards a general model of the multi-criteria workflow scheduling on the grid. *Future Generation Computer Systems*, 25(3):237 – 256, 2009.

[162] Mark Baker, Rajkumar Buyya, and Domenico Laforenza. Grids and Grid Technologies for Wide-Area Distributed Computing. *Software: Practice and Experience*, 32:1437 – 1466, 2002.

[163] M.L. Massie, B.N. Chun, and D.E. Culler. The Ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817 – 40, July 2004.

[164] Maui Scheduler Open Cluster Software. `http://mauischeduler.sourceforge.net`, August 2010.

[165] D.A. Menasce and E. Casalicchio. QoS in grid computing. *IEEE Internet Computing*, 8(4):85 – 7, July 2004.

[166] Aravind Menon, Jose Renato Santos, Yoshio Turner, G. Janakiraman, and Willy Zwaenepoel. Diagnosing performance overheads in the xen virtual machine environment. In *Proceedings of the First ACM/USENIX International Conference on Virual Execution Environments, VEE 05*, pages 13 – 23, Chicago, IL, United states, 2005.

[167] Microsoft Office 365. `http://www.microsoft.com/en-gb/office365/online-software.aspx`, September 2012.

[168] Microsoft SkyDrive. `http://windows.microsoft.com/en-US/skydrive/download-skydrive`, September 2012.

[169] MIKE2.0 Methodology. `http://mike2.openmethodology.org/`, September 2012.

[170] Minqi Zhou, Rong Zhang, Dadan Zeng, and Weining Qian. Services in the Cloud Computing era: a survey. In *Proceedings of 4th International Universal Communication Symposium (IUCS 2010)*, page 7, Piscataway, NJ, USA, 2010.

[171] G. Moltó and V. Hernández. On Demand Replication of WSRD-based Grid Services via Cloud Computing. In *Proceedings of the 9th International Meeting on High Performance Computing for Computational Science (VECPAR '10)*, pages 9 – 16, 2010.

[172] Jarek Nabrzyski, Jennifer M. Schopf, and Jan Weglarz, editors. *Grid resource management: state of the art and future trends*. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[173] Nadir K. Salih and Tianyi Zang. Survey and comparison for Open and closed sources in cloud computing. Technical report, School of Computer Science and Engineering, Harbin Institute of Technology, China, 2012.

[174] Nagios Monitoring System. `http://www.nagios.org/about/overview`, September 2012.

[175] Jun Nakajima and Mallick Asit K. Hybrid Virtualization - Enhanced Virtualization for Linux. In *Proceedings of the Linux Symposium*, 2007.

[176] R. Neumann, E. Goltzer, R. Dumke, and A. Schmietendorf. Caching Highly Compute-intensive Cloud Applications: An Approach to Balancing Cost with Performance. In *Joint Conf of 21st Int'l Workshop on Software Measurement and the 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA 2011)*, pages 96 – 105, Los Alamitos, CA, USA, 2011.

[177] Nimbus - An open source toolkit for Infrastructure-as-a-Service Cloud clusters. `http://workspace.globus.org/`, June 2012.

[178] Nimbus Context Agent. `http://www.nimbusproject.org/docs/current/clouds/clusters2.html#custom`, August 2012.

[179] Nimbus Context Broker. `http://www.nimbusproject.org/doc/ctxbroker/2.8/`, August 2012.

[180] National Institute of Standards and Technology (NIST). `http://www.nist.gov/`, September 2012.

[181] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. Eucalyptus: A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Useful Systems. Technical report, Computer Science Department University of California, Santa Barbara Santa Barbara, California 93106, October 2008.

[182] Open Cloud Consortium (OCC). `http://www.opencloudconsortium.org/`, September 2012.

[183] Open Grid Forum. `http://www.ogf.org/`, June 2012.

[184] The Object Management Group (OMG). /urlhttp://www.omg.org/, September 2012.

[185] OnLive - Video Games On Demand. `http://www.onlive.co.uk/`, September 2012.

[186] OpenCA Labs - Open Source Security and Identity Management Solutions. `http://www.openca.org/`, August 2012.

[187] OpenMP - An API specification for parallel programming. `http://openmp.org/wp/`, August 2012.

[188] OpenNebula - The Engine for Data Center Virtualization and Cloud Solutions. `http://www.opennebula.org`, June 2012.

[189] OpenStack: Open source software for building private and public clouds. `http://www.openstack.org/`, September 2012.

[190] OpenVZ. `http://wiki.openvz.org/Main_Page`, September 2012.

[191] OPTIMIS Toolkit. `http://www.optimis-project.eu/`, September 2012.

[192] Open Virtualization Format (OVF) - A standard from the Distributed Management Task Force. `http://www.dmtf.org/standards/ovf`, June 2012.

[193] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin. Performance Evaluation of Virtualization Technologies for Server Consolidation. Technical report, HP Labs, 2008.

[194] S. Patidar, D. Rane, and P. Jain. A survey paper on cloud computing. In *Proceedings of the 2012 Second International Conference on Advanced Computing & Communication Technologies (ACCT 2012)*, pages 394 – 8, Los Alamitos, CA, USA, 2012.

[195] V. Paxson and S. Floyd. Wide-area traffic: The failure of Poisson modeling. In *Computer Communication Review*, volume 24, pages 257 – 68, USA, 1994.

[196] Portable Batch System (PBS). `http://www.nas.nasa.gov/Software/PBS/pbsnashome.html`, June 2012.

[197] Phosphorus - QoS in scientific communities. `http://www.ist-phosphorus.eu`, June 2012.

[198] G.J. Popek and R.P. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412 – 21, July 1974.

[199] Puppet - IT Automation for System Administrators. `http://puppetlabs.com/`, August 2012.

[200] QEMU - Open Source Machine Emulation and Virtualizer. `http://www.qemu.org`, June 2012.

[201] Rackspace - The open Cloud company. `http://www.rackspace.co.uk`, September 2012.

[202] The Rackspace Cloud Powered By OpenStack. `http://www.rackspace.com/blog/next-generation-rackspace-cloud-servers/`, September 2012.

[203] L. Ramaswamy, Ling Liu, and A. Iyengar. Cache clouds: cooperative caching of dynamic documents in edge networks. In *25th IEEE International Conference on Distributed Computing Systems*, pages 229 – 38, Los Alamitos, CA, USA, 2005.

[204] Adit Ranadive, Mukil Kesavan, Ada Gavrilovska, and Karsten Schwan. Performance implications of virtualizing multicore cluster machines. In *HPCVirt '08: Proceedings of the 2nd workshop on System-level virtualization for high performance computing*, pages 1 – 8, New York, NY, USA, 2008. ACM.

[205] Red Hat. KVM - Kernel Based Virtual Machine. Whitepaper `http://www.redhat.com/resourcelibrary/whitepapers/doc-kvm`, 2009.

[206] N. Regola and J.-C. Ducom. Recommendations for Virtualization Technologies in High Performance Computing. In *Proceedings of the 2010 IEEE 2nd International Conference on Cloud Computing Technology and Science (CloudCom 2010)*, pages 409 – 16, Los Alamitos, CA, USA, 2010.

[207] Reservoir - Resources and Services Virtualization without Barriers. `http://www.reservoir-fp7.eu`, June 2012.

[208] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A taxonomy and survey of cloud computing systems. In *NCM 2009 - 5th International Joint Conference on INC, IMS, and IDC*, pages 44 – 51, Seoul, Korea, Republic of, 2009.

[209] B. Rochwerger, J. Caceres, R.S. Montero, D. Breitgand, E. Elmroth, A. Galis, E. Levy, I.M. Llorente, K. Nagin, and Y. Wolfstha. The Reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):Online, 2009.

[210] K. Roebuck. *Hypervisor: High Impact Technology - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors*. Emereo Pty Limited, 2011.

[211] Ron Yorston. Keeping Filesystem Images Sparse. `http://intgat.tigress.co.uk/rmy/uml/index.html`, July 2012.

[212] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[213] RDDTool. `http://oss.oetiker.ch/rrdtool/`, September 2012.

[214] GT 4.0 WS GRAM: RSL Job Description Schema Documentation. `http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/gram_job_description.html`, August 2012.

[215] MPICH2 : High-performance and Widely Portable MPI. `http://www.mcs.anl.gov/research/projects/mpich2/`, August 2012.

[216] Rusty Russell. Virtio: towards a de-facto standard for virtual I/O devices. *SIGOPS Oper. Syst. Rev.*, 42(5):95 – 103, 2008.

[217] A. Sahai, S. Graupner, V. Machiraju, and A. van Moorsel. Specifying and monitoring guarantees in commercial grids through SLA. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 292 – 299, May 2003.

[218] et al. Sakellariou, R. UNICORE: A Grid Computing Environment. In *Euro-Par 2001 Parallel Processing*, pages 825 – 834. Springer Berlin / Heidelberg, 2001.

[219] Mohsen Amini Salehi and Rajkumar Buyya. Adapting market-oriented scheduling policies for cloud computing. In *Proceedings of the 10th international conference on Algorithms and Architectures for Parallel Processing*, ICA3PP'10, pages 351 – 362, Berlin, Heidelberg, 2010. Springer-Verlag.

[220] SalesForce - The CRM Software as a Service (SaaS) Leader. `http://www.salesforce.com/`, June 2012.

[221] Gry Schneider, Hugo Kohmann, and Hakkon Bugge. Fault Tolerant Checkpointing Solution for Clusters and Grid Systems. Technical report, Toulouse-Oslo, 2007.

[222] Jennifer M. Schopf and Bill Nitzberg. Grids: The top ten questions. *Scientific Programming*, 10(2):103 – 111, 2002.

[223] S. Sharaf and K. Djemame. An Application of Dynamic Service Level Agreements in a Risk-Aware Grid Environment. In *2009 International Conference on Grid Computing and Applications (GCA'2009)*, Las Vegas, Nevada, July 2009.

[224] S. Sharaf and K. Djemame. Extending WS-Agreement to Support Re-Negotiation of Dynamic Grid SLAs. In *eChallenges 2010*, Warsaw, Poland, October 2010.

[225] Xuanhua Shi, Chao Liu, Song Wu, Hai Jin, Xiaoxin Wu, and Li Deng. A cloud service cache system based on memory template of virtual machine. In *6th Annual ChinaGrid Conference, ChinaGrid 2011*, pages 168 – 173, Dalian, Liaoning, China, 2011.

[226] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1 – 10, Washington, DC, USA, 2010. IEEE Computer Society.

[227] Cristina Simache and Marc Descamps. Fault Tolerant Storage Solution for Clusters and Grid Systems. Technical report, Toulouse, 2007.

[228] SimJava. `http://www.dcs.ed.ac.uk/home/hase/simjava/`, September 2012.

[229] Siva Theja Maguluri, R. Srikant, and Lei Ying. Stochastic models of load balancing and scheduling in cloud computing clusters. In *Proceedings of the IEEE INFOCOM*, pages 702 – 710, Orlando, FL, USA, March 2012. IEEE.

[230] SLA4DGrid - service-level agreements for the national D-Grid initiative. `http://www-ds.e-technik.uni-dortmund.de/~yahya/`, June 2012.

[231] SLA@SOI - Empowering the service industry with SLA-aware infrastructures. `http://sla-at-soi.eu/`, June 2012.

[232] Shava Smallen, Kate Ericson, Jim Hayes, and Catherine Olschanowsky. User-level grid monitoring with Inca 2. In *Proceedings of the 2007 Workshop on Grid Monitoring, GMW'07*, pages 29 – 38, Monterey, CA, United states, 2007.

[233] Storage Networking Industry Association (SNIA). `http://www.snia.org/`, September 2012.

[234] Biao Song, M.M. Hassan, Eui-Nam Huh, Chang-Woo Yoon, and Hyun-Woo Lee. A hybrid algorithm for partner selection in market oriented cloud computing. In *Proceedings - International Conference on Management and Service Science, MASS 2009*, Wuhan, China, 2009.

[235] Borja Sotomayor, Kate Keahey, and Ian Foster. Combining batch execution and leasing using virtual machines. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing 2008, HPDC'08*, pages 87 – 96, Boston, MA, United states, 2008.

[236] Borja Sotomayor, Rubé andn, Ignacio M. Llorente, and Ian Foster. Virtual Infrastructure Management in Private and Hybrid Clouds. *Internet Computing, IEEE*, 13(5):14 – 22, Sept 2009.

[237] A. Stage and T. Setzer. Network-aware migration control and scheduling of differentiated virtual machine workloads. In *2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD 2009)*, pages 9 – 14, Piscataway, NJ, USA, 2009.

[238] V. Stantchev. Performance evaluation of cloud computing offerings. In *Proceedings of the 2009 Third International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2009)*, pages 187 – 92, Piscataway, NJ, USA, 2009.

[239] Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, and Pat Helland. The end of an architectural era: (it's time for a complete rewrite). In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 1150 – 1160. VLDB Endowment, 2007.

[240] StreamMyGame. `http://www.streammygame.com/`, September 2012.

[241] A. Suciu and R. Potolea. A taxonomy for grid applications. In *IEEE International Conference on Automation, Quality and Testing, Robotics. (AQTR'08)*, volume 3, pages 365 –368, may 2008.

[242] Ajay Surie, H. Andrés Lagar Cavilla, Eyal de Lara, and M. Satyanarayanan. Low-bandwidth VM migration via opportunistic replay. In *HotMobile '08: Proceedings*

of the 9th workshop on Mobile computing systems and applications, pages 74 – 79, New York, NY, USA, 2008. ACM.

[243] M. Swany and R. Wolski. Representing Dynamic Performance Information in Grid Environments with the Network Weather Service. In *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*, pages 48–48, May 2002.

[244] A. Tchana, L. Broto, and D. Hagimont. Approaches to cloud computing fault tolerance. In *International Conference on Computer, Information and Telecommunication Systems (CITS 2012)*, pages 6 – 14, Piscataway, NJ, USA, 2012.

[245] Thomas Erl. The Principles Of Service Orientation. *Web Services Journal*, 5(9):10 – 18, September 2005.

[246] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter. The NetLogger methodology for high performance distributed systems performance analysis. In *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pages 260 – 267, July 1998.

[247] TORQUE Resource Manager. `http://www.adaptivecomputing.com/products/open-source/torque/`, August 2012.

[248] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, and D. Snelling. Open Grid Services Infrastructure (OGSI) Version 1.0, Global Grid Forum Draft Recommendation. `www.ggf.org/documents/GFD.15.pdf`, August 2012.

[249] J. Turnbull. *Pulling strings with puppet: configuration management made easy*. Springer, 2008.

[250] UNICORE. `http://www.unicore.eu/`, August 2012.

[251] W3C - World Wide Web Consortium. `http://www.w3.org/`, September 2012.

[252] G. Valle, T. Naughton, H. Ong, and S. L. Scott. Checkpoint/restart of virtual machines based on xen. In *High Availability and Performance Computing Workshop (HAPCW'06)*, Santa Fe, New Mexico, USA, 2006.

[253] Hien Nguyen Van, Frederic Dang Tran, and Jean-Marc Menaud. SLA-aware virtual resource management for cloud infrastructures. In *Proceedings - IEEE 9th*

International Conference on Computer and Information Technology, CIT 2009, volume 1, pages 357 – 362, Xiamen, China, 2009.

[254] Luis M. Vaquero, Luis Rodero Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50 – 55, 2009.

[255] VDI - Virtual Disk Image. `http://www.virtualbox.org/manual/ch05.html`, June 2012.

[256] Microsoft VHD - Virtual Hard Disk Image Format. `http://technet.microsoft.com/en-us/virtualserver/bb676673.aspx`, June 2012.

[257] QCOW2 - Image Format. `http://people.gnome.org/~markmc/qcow-image-format.html`, June 2012.

[258] VirtualBox. `http://www.virtualbox.org/`, September 2012.

[259] VMDK - Virtual Machine Disk Format. `http://www.vmware.com/technical-resources/interfaces/vmdk.html`, June 2012.

[260] VMware. `http://www.vmware.com/`, September 2012.

[261] VMware ESX Hypervisor. `http://www.vmware.com/`, June 2012.

[262] VMware Infrastructure: Resource Management with VMware DRS. White Paper `http://www.vmware.com/pdf/vmware_drs_wp.pdf`, 2006.

[263] VMware Server. `http://www.vmware.com/products/server/overview.html`, September 2012.

[264] VMware vCloud Suite. `http://www.vmware.com/products/datacenter-virtualization/vcloud-suite/overview.html`, September 2012.

[265] VMware Workstation. `http://www.vmware.com/products/workstation/overview.html`, September 2012.

[266] M.A. Vouk. Cloud computing - Issues, research and implementations. In *2008 30th International Conference on Information Technology Interfaces (ITI)*, pages 31 – 40, Piscataway, NJ, USA, 2008.

[267] Linux VServer. `http://linux-vserver.org`, September 2012.

[268] VMware vSphere. `http://www.vmware.com/products/datacenter-virtualization/vsphere/index.html`, September 2012.

[269] W3C. Simple Object Access Protocol (SOAP) 1.2. `http://www.w3.org/TR/soap/`, April 2007.

[270] Feng Wang, Jie Qiu, Jie Yang, Bo Dong, Xinhui Li, and Ying Li. Hadoop high availability through metadata replication. In *International Conference on Information and Knowledge Management, Proceedings*, pages 37 – 44, Hong Kong, China, 2009.

[271] W3C Web Service Definition. `http://www.w3.org/TR/ws-arch/#whatis`, September 2012.

[272] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[273] V. Winkler. *Securing the Cloud: Cloud Computer Security Techniques and Tactics*. Elsevier Science, 2011.

[274] Timothy Wood, K. K. Ramakrishnan, Prashant Shenoy, and Jacobus van der Merwe. CloudNet: dynamic pooling of cloud resources by live WAN migration of virtual machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 121–132. ACM, 2011.

[275] Web Services Description Language (WSDL) Specification 1.1. `http://www.w3.org/TR/wsdl`, August 2012.

[276] Web Service Resource Framework - The WS-Resource Framework. `http://www.globus.org/wsrf/`, June 2012.

[277] Xen 4.1 Release Notes. `http://wiki.xen.org/wiki/Xen_4.1_Release_Notes`, September 2012.

[278] Xianghua Xu, Feng Zhou, Jian Wan, and Yucheng Jiang. Quantifying performance properties of virtual machine. In *2008 International Symposium on Information Science and Engineering (ISISE)*, volume 1, pages 24 – 8, Piscataway, NJ, USA, 2008.

[279] Xiaofei Zhang and Lei Chen. Fault tolerance study for durable storage on the cloud. In *International Conference on Cloud and Service Computing (CSC 2011)*, pages 360 – 5, Piscataway, NJ, USA, 2011.

[280] Kaiqi Xiong and Harry Perros. Service performance and analysis in cloud computing. In *SERVICES 2009 - 5th 2009 World Congress on Services*, pages 693 – 700, Bangalore, India, 2009.

[281] Cong Xu, Yuebin Bai, and Cheng Luo. Performance evaluation of parallel programming in virtual machine environment. In *NPC 2009 - 6th International Conference on Network and Parallel Computing*, pages 140 – 147, Gold Coast, QLD, Australia, 2009.

[282] Zhao Yi and Huang Wenlong. Adaptive distributed load balancing algorithm based on live migration of virtual machines in cloud. In *NCM 2009 - 5th International Joint Conference on INC, IMS, and IDC*, pages 170 – 175, Seoul, Korea, Republic of, 2009.

[283] Nezih Yigitbasi, Alexandru Iosup, Dick Epema, and Simon Ostermann. C-Meter: A framework for performance analysis of computing clouds. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID 2009*, pages 472 – 477, Shanghai, China, 2009.

[284] Jia Yu, Rajkumar Buyya, and Kotagiri Ramamohanarao. Workflow scheduling algorithms for Grid computing. In *Metaheuristics for Scheduling in Distributed Computing Environments*, pages 173 – 214. Springer, 2008.

[285] Shen Yu Liang and Xu Lu. An efficient disk I/O characteristics collection method based on virtual machine technology. In *Proceedings - 10th IEEE International Conference on High Performance Computing and Communications, HPCC 2008*, pages 943 – 949, Dalian, China, 2008.

[286] Hai Zhong, Kun Tao, and Xuejie Zhang. An Approach to Optimized Resource Scheduling Algorithm for Open-Source Cloud Systems. In *Proceedings of the The Fifth Annual ChinaGrid Conference*, CHINAGRID '10, pages 124 – 129, Washington, DC, USA, 2010. IEEE Computer Society.

Appendix A

Contextualization Tools Class Diagram

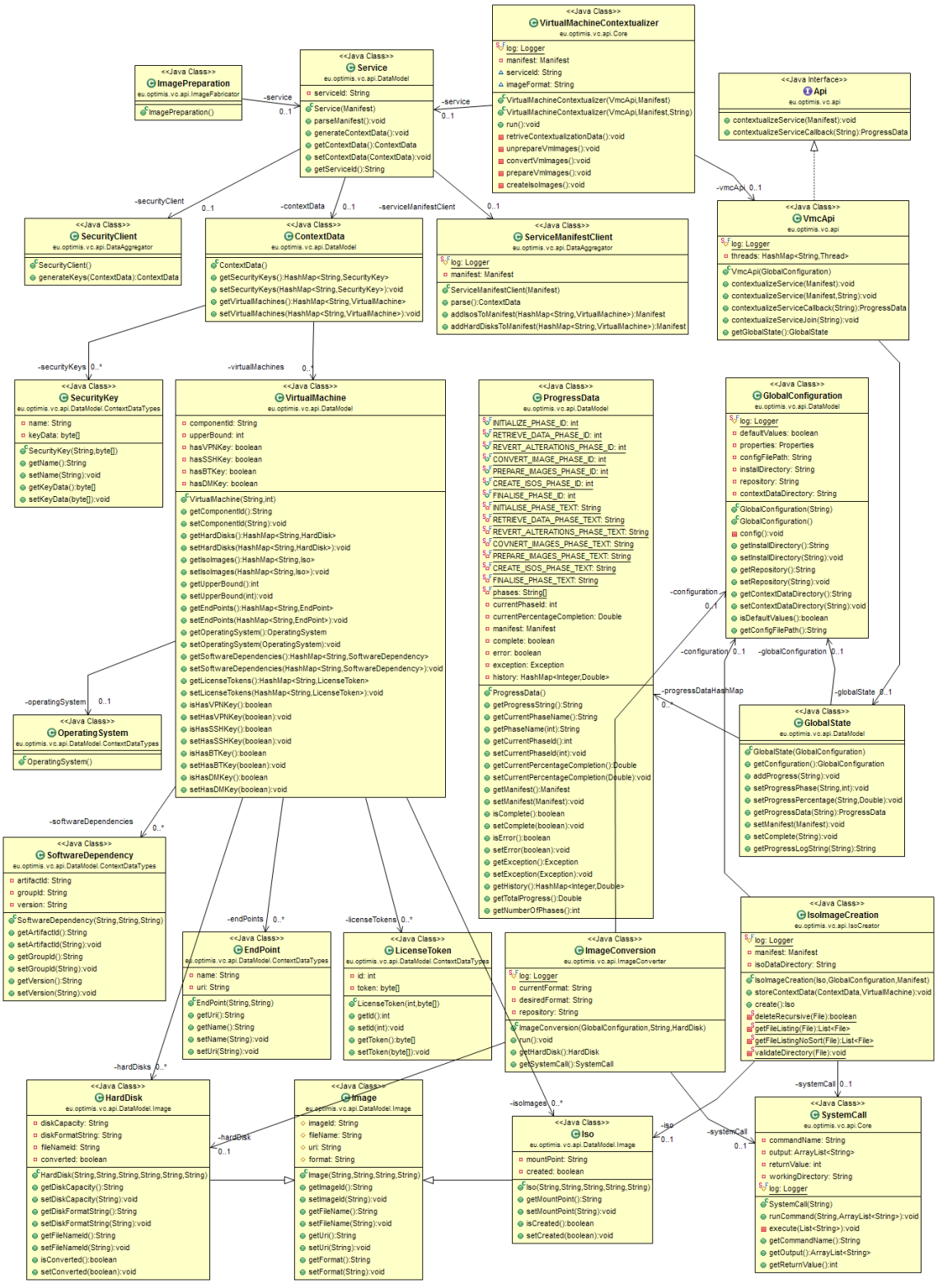


Figure A.1. Overview Class Diagram of the Contextualization Tools.