Georg-August-Universität
Göttingen
Zentrum für Informatik

ISSN 1612-6793
Nummer ZFI-MSC-2010-

Masterarbeit

im Studiengang "Angewandte Informatik"

Extension of a Globus Toolkit 4 Grid System by a Virtual Runtime Environment based on Eucalyptus

Dalia Dahman, geb. Sarsour

am Institut für

Informatik
Gruppe Softwaretechnik für Verteilte Systeme

Bachelor- und Masterarbeiten
des Zentrums für Informatik
an der Georg-August-Universität Göttingen

17. September 2010

Georg-August-Universität Göttingen
Zentrum für Informatik

Goldschmidtstraße 7
37077 Göttingen
Germany

Tel. +49 (5 51) 39-17 42010
Fax +49 (5 51) 39-1 44 15
Email office@informatik.uni-goettingen.de
WWW www.informatik.uni-goettingen.de

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Göttingen, den 17. September 2010

Master's thesis

Extension of a Globus Toolkit 4 Grid System by a Virtual Runtime Environment based on Eucalyptus

Dalia Dahman, née Sarsour

September 17 th 2010

Supervised by
Prof. Dr. Jens Grabowski
Software Engineering for Distributed Systems Group
Georg-August-Universität Göttingen

Abstract

Currently, the need for a powerful and scalable infrastructure that resolves computational and data intensive tasks and provides on-demand scalable resources increases rapidly. We think that the integration of existing technologies, such as grid and cloud infrastructures instead of building a new infrastructure can fulfill this demand.

This thesis is concerned with the integration of such infrastructures. It investigates possible and relevant consolidation scenarios of grid and cloud systems. Possible interoperability approaches that can be used to realize any of these consolidation scenarios are presented and discussed. One of these consolidation scenarios is the extension of an existing grid system with resources provided by a cloud infrastructure. This scenario is implemented as a case study for the integration of Globus Toolkit 4 grid system and Eucalyptus cloud infrastructure. Different design scenarios for extending Globus Toolkit 4 grid system with a Virtual Runtime Environment provided by Eucalyptus framework are introduced and discussed. A description of the implementation of the selected design scenario is provided. We show the successful deployment of the case study. This includes the execution of an application on grid resources as well as on Virtual Runtime Environment provided by Eucalyptus framework.

Contents

1 Introduction

As a result of the advances in different scientific domains and the increasing need for resources that are able to resolve computational intensive tasks, a large computation and processing power over large data sets are demanded. Therefore, the concept of grid computing was initiated in the late 1990s to utilize geographically-distributed heterogeneous computational and storage resources. These resources are harnessed by grid to solve large-scale problems and to achieve better performance and throughput.Recently, the cloud computing paradigm has been developed to provide businesses and users with on-demand scalable resources such as networks, servers, storage, applications, and services.

Nowadays, researches have become more concerned with the relation between grid and cloud computing paradigms. Some researches see them in competition, while others insist that they complete each other. The similarity of the goals of both technologies is the origin of such a confusion about the relationship between grid and cloud technologies. For example, minimizing the computation costs is a common goal of both technologies.

At present, cloud computing infrastructure offer only services based on Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) service models and within one domain. Furthermore, users are responsible for the management of these services and resources provided by cloud infrastructure. However, grid computing is concerned with different issues, such as federated interoperation of computing facilities, security, shared data management, system monitoring, and application scheduling and execution.

The need for a powerful and scalable infrastructure that resolves scientific computational and data intensive tasks and provides on-demand scalable resources is increasing rapidly. Therefore, we think of the combination of existing technologies such as grid and cloud infrastructures instead of developing a new infrastructure can fulfill such a demand and provides better services.

1.1 Motivation

A hybrid platform combining grid and cloud technologies would provide benefits to resource providers, as well as to end users. By using cloud technologies, grid systems can mitigate their limitations. To fulfill different and several users demands, grid systems need to provide dynamically scalable resources and services. Furthermore, the develop-

ment and the deployment of grid-based applications face challenges of working over large sets of heterogeneous resources, which belong to different organizations and include different access levels. By applying cloud technologies including virtualization on the top of grid resources, a level of homogenization can be achieved, which simplifies resources management. In addition, grids can gain benefits from cloud technologies via a virtualization layer, which enables users to customize their environments in order to avoid failure from misconfigured systems or from insufficient allocation of disk space and memory. Cloud systems can also benefit from grid features and services, such as security, shared data management, system monitoring, and application or job scheduling and execution.

1.2 Goal Description

This thesis investigates the possible consolidation scenarios of grid and cloud technologies according to user needs. It also examines interoperability approaches between grids and clouds. It concentrates on providing services offered by cloud technologies to grid users. The extension of a Globus Toolkit 4 (GT4) [44] grid system with resources provided by Elastic Utility Computing Architecture Linking Your Programs to Useful Systems (Eucalyptus) [5] cloud technology, a consolidation scenario is carried out as a case study in this thesis. Possible design scenarios of the extension are investigated. In order to validate the proposed design, we show the execution of an application on grid resources, as well as on a Virtual Runtime Environment (VRE) provided by Eucalyptus framework.

1.3 Thesis Structure

This thesis is structured as follows: Chapter 2 describes the underlying technologies covered for the achievement of the thesis goals. It presents mainly the cluster computing paradigm, the grid computing paradigm, virtualization technology concepts, and the cloud computing paradigm. Chapter 3 explains the consolidation of grid and cloud systems. This includes possible integration scenarios of grid and cloud systems and interoperability approaches between them. The possible scenarios for extending a GT4 grid system with a VRE provided by Eucalyptus framework are introduced in Chapter 4. Chapter 5 describes the realization of the thesis goal by implementing a case study. This includes an overview of the system and the environment setup. It also describes how grid users can use a VRE to execute their jobs. Chapter 6 concludes this work by providing a summary and outlook.

2 Foundations

This chapter introduces the underlying technologies serving the main goal of this thesis. Section 2.1 presents the cluster computing paradigm. The grid computing paradigm and the GT4 middleware are depicted in Section 2.2. In addition, Section 2.3 describes virtualization technology and its concepts. The cloud computing paradigm and its services are presented in Section 2.4.

2.1 Cluster Computing

In the 1980s, the cluster computing approach gains momentum by merging high-performance microprocessors, high-speed networks and standard tools for high performance distributed computing. In addition, the need of processing power for computational intensive applications raises. Furthermore, the high cost and lack of availability of traditional supercomputers force researchers to find alternative solutions, where the production and development of processors are limited by light speed, thermodynamic laws and costs.

The recent progress and availability of high-speed networks and high-performance microprocessors make cluster computing a good solution, where using parallel architectures is much cheaper than the use of sequential supercomputing and achieves equivalent performance level as these supercomputers [61, 78].

Pfister's [61] and Buyya's [39] work defines clusters as follows: "A cluster is a type of parallel or distributed computer system, which consists of a collection of interconnected stand-alone computers working together as a single integrated computing resource".

Typically, cluster resources are homogeneous, are connected via Local Area Network (LAN), locate in a single administrative domain and are managed by a single entity. Furthermore, clusters are built by using off-the-shelf hardware components and commonly used software to offer parallel-distributed platforms for high-performance, high-throughput and high-availability computing.

The common architecture of a cluster is shown in Figure 2.1. The main components of a cluster are the standalone computers (e.g. Personal Computer (PC)s or Workstations), operating systems, high-performance networks, parallel programming environments (e.g. compilers), applications (e.g. system management tools) and the middleware, which consists of two layers (Single System Image (SSI) and Availability Infrastructure). SSI represents the distributed system as a single unified computing resource. The availability

infrastructure provides all nodes of the cluster with cluster fault-tolerant services such as the automatic recovery from failures [34].

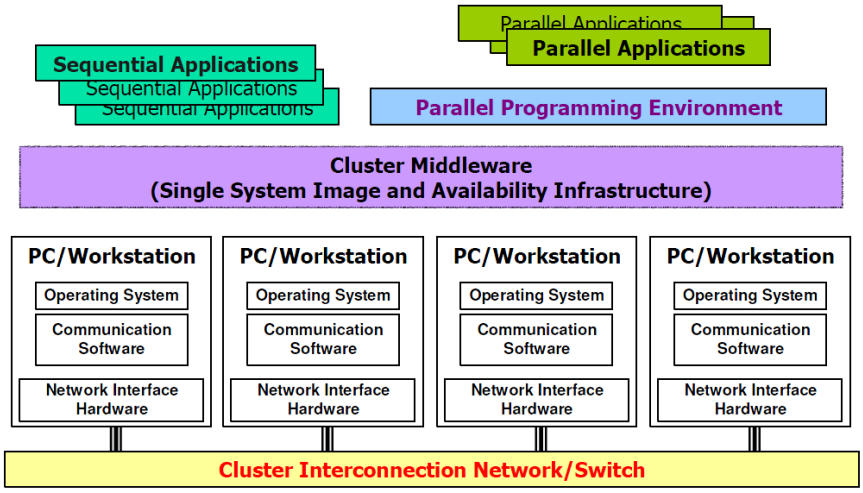

Figure 2.1: Cluster architecture (R. Buyya [39])

2.1.1 Resource Management System Middleware

A Resource Management System (RMS) performs as a cluster middleware that implements the SSI [40] for the underlying distributed system. Via RMS, users can execute their jobs on the cluster without the need to understand the complexities of cluster architecture. In addition, it manages the cluster through four capabilities, i.e., resource management, job queuing, job scheduling, and job management.

The basic RMS architecture is built using the client-server paradigm [34], where the client daemons (i.e., service requester) submit tasks and the server daemon distributes tasks between resources (i.e., service providers). These daemons maintain tables, which store information about the whole underlying distributed system. Once a job has been submitted to the RMS, the job is placed, scheduled and executed according to the submitted requirement. A *job* [36] is defined as, it is the main execution object, which consists of a collection of related processes, that it is managed as a one unit. The life cycle of a *job* can be divided into four states. The states are creation, submission, execution and finalization (e.g. retuning results) states. At creation state a script contains all of the parameters of the *job* is prepared.

Figure 2.2 represents a common architecture of cluster RMS [78]. A RMS manages resources such as processors and disk storage in the cluster. Furthermore, it maintains status information of resources. Therefore, jobs can be assigned to available machines. The

RMS software on server node consists of two components a resource manager and a job scheduler, where the resource manager is responsible for locating and allocating resources, authentication, as well as job execution and migration. The job scheduler component is concerned with specifying which execution nodes to use and when to start the job.
The users submit jobs to the RMS, where they are hold by queues until there are available resources. Afterwards, the RMS invokes a job scheduler to choose one of the jobs from queues to be executed on available machines. Then it becomes responsible for managing job execution and returning the results to users.

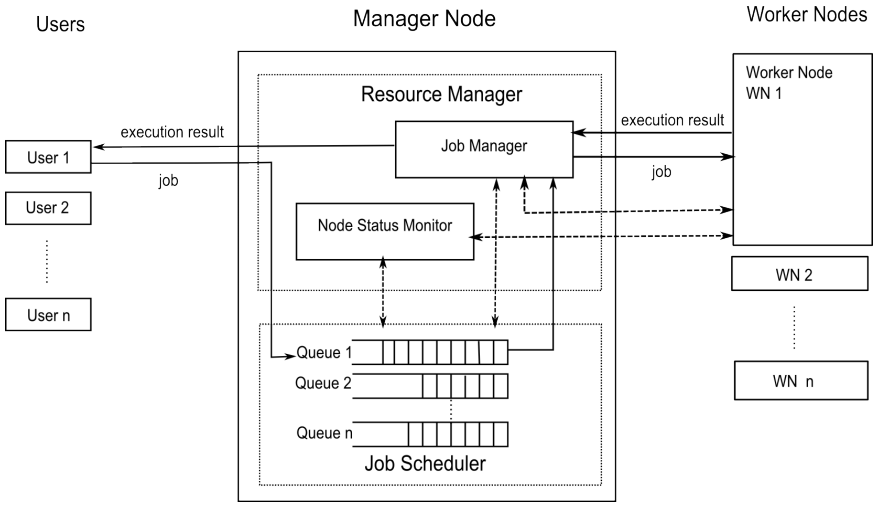

Figure 2.2: Cluster RMS architecture [78]

The Portable Batch System (PBS) [36], CONDOR [59] and Load Share Facility (LSF) [41] are leading examples of implemented RMS middleware.

2.1.2 Portable Batch System

PBS [36] is a widely used cluster RMS software package and a resource management system. It was developed by the National Aeronautics and Space Administration (NASA) [56] to fit to the Portable Operating System Interface for UNIX (POSIX) batch environment standards . PBS has a flexible scheduler module that can be replaced by another to enable sites of selecting the more appropriate scheduling policies. PBS software consists of the following components:

- The **pbs_server** acts as the resource manager (see Figure 2.2), which creates and modifies information about the jobs and the resources. It is responsible for failure tolerance by storing a copy of completed jobs in the backing store.

- The **pbs_sched** is the job scheduler, where it makes scheduling decision depending on information submitted with the job. Usually **pbs_server** and **pbs_sched** run on the same machine.
- Submit nodes (**pbs_client**) which have client interfaces that enable users to submit and track their jobs.
- Worker nodes (**pbs_mom**[1]), where their main task is to manage and execute submitted jobs. Furthermore, **pbs_mom** communicates with **pbs_server** on the manager node and it delivers the result to the client.

The pbs_sched job scheduler can be replaced with the Maui scheduler[63], which is an open source and external job scheduler designed to work with resource management system such as PBS. In comparison with pbs_sched scheduler, the Maui scheduler supports powerful scheduling policies described in the following:

- Reservation is a method of allocation resources for specific time. This policy has an enforcement nature.
- Fairshare gives equal priority to each one of the jobs in a queue.
- Backfill allows the scheduler to run low priority jobs on the available nodes without delaying the higher priority jobs.
- Throttling Policies control resource consumption in order not to make the system being dominated by individual user, group, account, Quality of Service (QoS) by limiting the total number of jobs assigned to each of them.

2.2 Grid Computing

As a consequence of the scientific advances in different domains, such as unlocking the genetic code, exploring the structure of the matter and the universe as well as the environmental simulations, the need of short-term and very large computational intensive power increases. Furthermore, the vast data generated and analyzed within e-Science programs would exceed the capacity of any centralized storage system. For example, CERN Large Hadron Collider (LHC) [25] is a particle accelerator, that can generate about 10 Petabytes of information a year. Performing such data intensive and computationally intensive tasks requires highly expensive resources (supercomputers), which typically could not be afforded. Such challenges motivate the creation of a powerful, inexpensive and flexible computational infrastructure -known as grid computing- by joining geographically-distributed

[1] Machine Oriented Miniserver (MOM)

heterogeneous computational and storage resources such as databases, storage servers, supercomputers and clusters for solving large-scale problems and achieving greater performance and throughput by coupling resources from different distributed organizations. Before grid computing each organization could only use the resources, over which it has direct control [37, 68].

Foster [45] defines grid system in three check points as followed:

A grid is a system that:

- coordinates resources that are not subject to centralized control by integrating resources from different administrative domains, and addressing security, membership and policy.
- uses standard, open, general-purpose protocols and interfaces to address authentication, authorization, resources discovery/access, etc. Protocols/interfaces must be standard and open (interoperable).
- then delivers nontrivial qualities of service by using its software and hardware resources in such a way they are capable of delivering various QoSs to meet complex user requirements.

2.2.1 A Grid System taxonomy

Grid systems can be classified into three types as shown in Figure 2.3: 1) computational grid [47], 2) data grid [73], and 3) service grid. Computational grids offer high throughput and reliable distributed supercomputing capabilities. Data grids enable the combination of the existing data scattered in a wide area networks, and make these data available as digital libraries and data warehouses. The service grids offers services which are not offered by any single resource. Service grids are divided into three types, i.e., on-demand, collaborative, and multimedia [65].

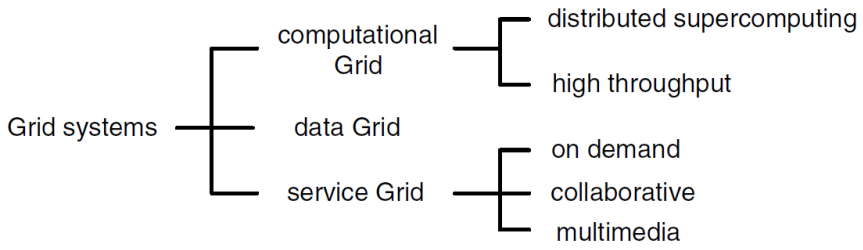

Figure 2.3: A grid systems taxonomy [54]

2.2.2 Grid Architecture

A general grid architecture has been defined by Foster [48]. It emphasizes that the components of grid are classified according to functionality and purpose. Figure 2.4 gives an overview of the grid architecture in terms of layer by analogy to the Internet Architecture. A description of each layer is summarized in the following [48]:

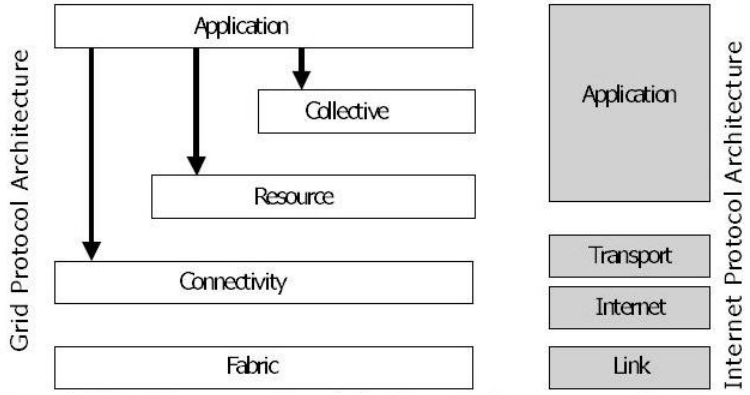

Figure 2.4: Layered Grid Architecture [48]

- Fabric: represents the sharing of resources in the grid such as computers, cluster and storage.
- Connectivity: represents the protocols that allow the communication among the resources such as Transmission Control Protocol/Internet Protocol (TCP/IP), X509 certificates and Grid Security Infrastructure (GSI). Furthermore, this layer includes additional security protocols that emerged in grid community in response the security challenges.
- Resources: represents all the protocols and services needed for the individual resource management. For instance, it enables the initiation, monitoring, and accounting of resources. There are two classes of protocols in this layer:
  - Information Protocols: Allow accessing the information of resources such as Central Processing Unit load in the case of computers.
  - Management Protocols: Allow the control of resources to a specific degree. The degree of control corresponds to the organization's policies.
- Collective: represents the set of protocols and services which enables the management of multiple resources. The services of this layer are built on the services of the

resource layer to allow taking a collection of resources that work together to solve some common task. Samples of the services found in this layer are:

  - Resource Registry: discover resources in a Virtual Organization (VO) and their properties.
  - Allocation and scheduling of services: allocate resources to jobs instead of the random selection. For this purpose a scheduler decides what jobs run where and when.
  - Monitoring services: monitor the proper work of all resources.
  - Data management services: keep track of datasets which are required for the processing of jobs, and transfer them to the resource that needs them.

- Application: refers to the real applications that run on a grid. These applications can interact with the lower layers and call their services.

2.2.3 Open Grid Service Architecture

Grid systems should enable the dynamic integration, virtualization and management of their resources and services, in spite of the heterogeneity and geographical-distribution of them [48]. Therefore, it became clear that the services provided by grid systems need to be independent of implementations and applications. Standardization is the solution to enable the discovery, access, allocation and monitoring of grid environment [46].

For this task, researchers and users in the Global Grid Forum (GGF) [23] met to define grid specifications that result in adaption of standards and interoperable applications.

Open Grid Service Architecture (OGSA) [46] is developed by the GGF to address the interoperability between grids by merging grid and web service technologies in order to create sophisticated distributed systems. These grids could be built by different toolkits. OGSA is an extension of a set of services which are composed in different ways to satisfy the needs of VOs which operates them.

OGSA developers aimed at defining the core loosely-coupled services, their interfaces, and the semantics, behavior and interaction of these services, while they did not specify programming language or execution environment. These services are: execution management services (e.g. Job manager and execution planning services), data services (e.g. Data storage, access and transfer), resource management service, security services (e.g. Authorization and authentication) and information service (e.g. Discovery and monitoring). Furthermore, OGSA represents standard mechanisms for creating and naming grid service instances [46].

2.2.4 Web Services Resource Framework

The Web Services Resource Framework (WSRF) [12] is a combined effort of both Grid and Web services communities to provide a standard infrastructure for grid environments. The Web service term describes an important emerging distributed computing paradigm Business-to-business (B2B) that differs from other approaches such as CORBA, and Java RMI. Web services depend on open standards to ensure the interoperability between different software applications, running on heterogeneous platforms.

Grid services can be defined with Web service description language (WSDL), communicate with each other using Simple Object Access Protocol (SOAP) messages and use web services security layer. However, grid services disagree with typical web services in two special characteristics, that they are often transient and stateful. The Implementation of such stateful services is based on the WSRF standard.

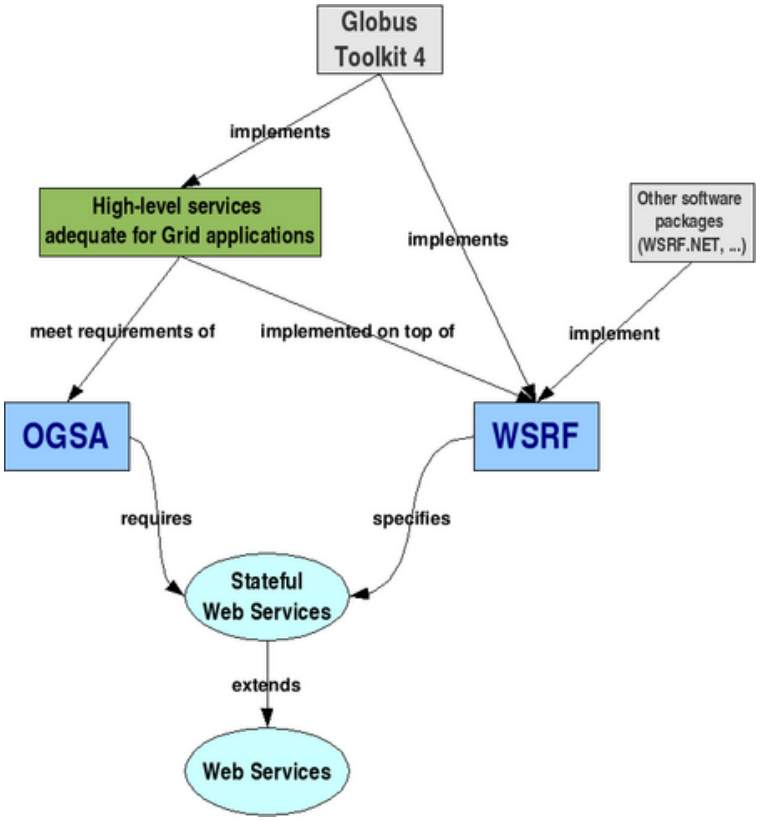

Figure 2.5: Relationship between OGSA, WSRF, GT4 and Web Services [68]

2.2.5 Globus Toolkit 4

"The Globus Toolkit 4 (GT4) is a software toolkit -open source-, developed by the Globus Alliance [44]. which can be used to create grid systems" [68]. The relation between OGSA and WSRF and how they relate the GT4 and Web services is shown in Figure 2.5, where OGSA standardizes all the services, that can be found in a grid system, while WSRF specifies and provides the stateful services that OGSA needs. In other words, OGSA is the architecture, while WSRF is the infrastructure on which that architecture is based. In addition, GT4 includes a few of high-level services that meet the requirements of OGSA and can be used to build Grid applications. Therefore, it includes resource monitoring, service discovery, job submission, security, and data management services. GT4 implements most of these services on top of WSRF, in addition to other services that are not implemented on to of WSRF and are called non-WS components [44].

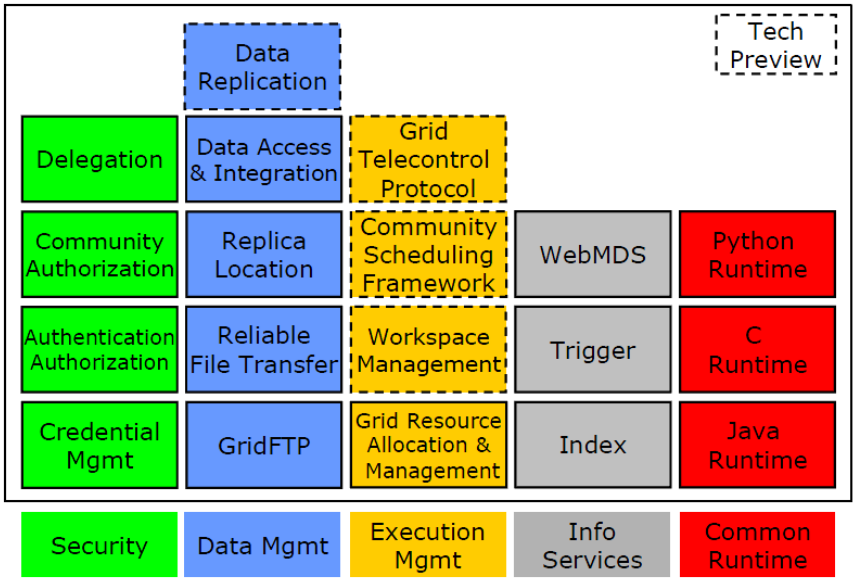

Figure 2.6: Globus Toolkit 4 (GT4) components [44]

The GT4 is organized as a set of loosely coupled components. These components are composed of services, programming libraries and development tools for building grid applications. The GT4 components [44] cover five wide domain areas, are shown in Figure 2.6 and explained in the following:

- Security components: refer to the GSI which facilitates the secure communications and applications. They include authentication and authorization, delegation, community authorization, and credential management.

- Data Management components: enable the discovery, transfer, and access of data. They include GridFTP, Reliable Transfer Service (RFT), Replica Location, Data Replication, and Open Grid Services Architecture Data Access and Integration (OGSA-DAI).
- Execution Management components: handle the deployment, scheduling, and monitoring program execution, i.e. jobs. They include Grid Resource Allocation and Management (GRAM), Community Scheduler Framework (CSF), Workspace Management, and the Grid Telecontrol Protocol.
- Information Services: monitor, index and discover the resources in Virtual Organization (VO). They include Index Service, Trigger Service, and WebMDS.
- Common runtime components: provide libraries and tools to host existing services and develop new ones. They include C Runtime, Java Runtime, and Python Runtime.

Components marked as "Tech previews" in Figure 2.6 are not well tested as other components and may change in the future.

2.2.6 Gridification

The grid community certainly aims and attempts to simplify and accelerate the development of grid applications. However, it sounds better to transform conventional applications (e.g. desktop applications and legacy applications) to be able to work on grid infrastructures, in order to benefit from grid features, such as computational intensive power and massive storage resources. In addition, this adaptation could save the costs and effort of software development [55].

"Gridification" is the process of adapting an existing application to be executed on grid environments. This process involves handling not only the execution of an application but also the execution speed-up through the acquisition of resources from outside, which were not accessible before [67].

The main idea of **gridification process** the is the parallelization of the task by dividing it into several smaller jobs [67]. The gridification process involves a management system and grid middleware services, i.e., components.

The division process is carried out by a management system. This system determines all jobs and their descriptions is the management system responsibility and the synchronization of the jobs as well. The management system should guarantee the correct and the right order of execution of the jobs considering the job dependencies. These dependencies have to be entered by user and the rules of the division of the task into smaller jobs as well. The entry of these data can be hard and time-consuming, because of the necessity

of management system readapting for similar applications or new management system implementing.

After the determination of the jobs and their order according to their dependencies the management system uses the services of the grid middleware. For job submission the execution service of the grid middleware is used. Each grid middleware usually has a command line tool to call its services. This command line calls includes the job description file which contains for instance the application name, the input data etc.

In addition, the management system inquires the status of the jobs (e.g. running, finished or failed) using the monitoring services provided by grid middleware. Then it collects the output data from finished jobs via grid data and execution services. The collection of data means copying files or transferring data from nodes [65, 67].

The gridification of an application may face some difficulties, which may make the applaying of the process is infeasible. The application may be restricted by a license agreement. Furthermore, the application can have hidden security weaknesses, which will be very dangerous in case of remote grid job execution. In addition, some applications use non-standard communication protocols or may depend on some libraries or executables which are not available by default on worker nodes. In some cases, there is a need to change the source code of the application.

The gridification process [74] should be applied on computationally intensive, data intensive and time-consuming applications. Such applications need many calculations, long run-time as well as huge disk space. Furthermore, the application should have a single interface to be gridified, from which the user can invoke the main function, specify input and configuration parameters, and get the output data files. The functionality of an application is known by users, but often they have no idea about the implementation. In other words, the application is considered as a black box, where the users are concerned only with the input and the output parameters.

The deployment of applications on the grid can be derived in a static or a dynamic manner. They can be pre-installed on grid machines by the system administrators. Otherwise, the applications can be wrapped in compiled executables, which should be submitted at run-time to grid machines with jobs.

2.3 Virtualization

In the 1960s, virtualization concept was first introduced by IBM to allow logically partitioning of large mainframe hardware. These partitions allow mainframe computers to run several tasks simultaneously. By the 1990s, virtualization was used mainly to recreate end user environments on a mainframe computer. For example if the IT administrator wants to introduce a new software and check its workability on different operating systems, he could apply virtualization technologies to create these different environments [71].

As Susanta [71] defines virtualization as follows: "Virtualization is a technology that combines or divides computing resources to present one or many operating environments using methodologies like hardware and software partitioning or aggregation, partial or complete machine simulation, emulation, time-sharing, and others". The virtualization provides a logical view of resources instead of the physical view. For example, it splits up a single physical server into multiple logical ones, where each logical server can run an operating system and applications independently [28].

The use of virtualization technologies has many advantages but also disadvantages. Some arguments for the use of virtualization technologies are listed in the following [71]:

- **Server Consolidation:** increases utilization rates, due to the run of multiple Virtual Machines (VMs) on one physical hardware. As a result of server consolidation, costs are saved and the administration costs is reduced.
- **Support Legacy applications:** legacy applications can continue to run on old Operating System s (OSs) that run as a guest operating systems on VMs.
- **Testing and development:** by using VMs the deployment and development of applications could be accelerated because of the isolation in known and controlled environments. Unknown factors such as mixed libraries caused by numerous installs can be eliminated. Severe crashes that required hours of reinstallation take moments by copying a preconfigured virtual image.
- **Improving System Reliability and Security:** The virtualization technologies encapsulates each guest operating system in a VM, which offers a higher reliability: If one VM crashed, the other VMs and the host system will not be affected.
- **Simplify Deployment:** the deployment of a new server, usually needs the hardware must be acquired, the software to be installed and then the new server must be connected to the network. But creating a new VM is possible within minutes, VMs can also be copied or moved to another host.

Some disadvantages of virtualization are licensing issues, virtual server sprawl, network complexity issues, hardware start-up costs, and failover costs. In the following, a brief descriptions of the disadvantages of virtualization are introduced [51, 38]:

- **Licencing issues:** virtualization enables several instances of software to run concurrently on one location (e.g. server). This fact enlarges the possibility of license violations. Software is implicitly created or moved at each time a VM is created or moved.
- **Virtual server sprawl:** is a situation when the number of VMs on a network reaches a point where the administrator can no longer manage them effectively. Virtual server

sprawl emphasises on management rather than physical proliferation, which is the case in the traditional server model.

- **Network complexity:** VMs require the same networking requirements as the physical servers. For example, twenty VMs running on two servers require the same number of Virtual LANS (VLANS) as twenty physical servers. Another problem of virtual machines is, the fact that the failure of a physical server, will cause the failure of all hosted VMs by it as well.
- **Hardware costs:** the reduction of the hardware in virtualization does not necessarily result in cost savings, since virtualization requires a high end server for supporting virtualization. Such type of high end and multicore servers is more expensive than midrange dualcore servers.
- **Failover costs:** costs of a failure in virtualization is very high compared with failure of a physical server. For example, if ten physical servers are consolidated into one single virtual server, which is subject to fail. The consequences influence the ten physical servers. Manipulating such a problem leads to additional high costs.

2.3.1 Virtual Machine Monitor (Hypervisor)

Virtualization technologies abstract the underlying hardware for VMs. The Virtual Machine Monitor (VMM) (also known as hypervisor) creates interfaces, which represent the virtualized resources such as Central Processing Units (CPUs), physical memory, network connections, and block devices. The hypervisor offers secure environments to meet the requirements of resources in recent computing systems [32].

The main approaches for virtualization implementation are:

- **Full virtualization:** provides a total abstraction of the physical resources and creates a complete virtual system in which the guest operating systems can run. There is no need to modify the guest OS or the applications. This approach enables the complete decoupling of the OS from the hardware. As a consequence, the migration of workloads between different physical systems is possible. In addition, the different applications could be isolated completely, which make this approach highly secure. However, the full virtualization may cause a decline in performance. An image of the whole system must be provided to the VMs, including virtual BIOS, virtual memory space, and virtual devices [32].
- **Para-virtualization:** provides an abstraction of the physical resources, which is similar but not identical to the underlying hardware. This approach provides higher performance than the full virtualization approach, although the host operating systems require modifications to be able to run as VMs on Xen [31]. However, there is no

need to apply any modifications on the Application Binary Interface (ABI). Existing user applications can run without any modification [32].

2.3.2 Xen Virtualization Technology

Xen [31] is a widely-used high performance hypervisor developed at the University of Cambridge. It an open source software based on para-virtualization technology [35].

The Xen-based system architecture [52] is illustrated in Figure 2.7 and described in the following:

- **Xen hypervisor VMM:** is above the hardware layer and has a direct access to it, which means that Xen hypervisor runs in the most privileged processorlevel. Then, the Xen domains VMs are running at the top of Xen hypervisor.
- **Domain0 (Dom0):** is starting at boot time, and is permitted to access Xen control interface. It administrates the creation, termination, and migration of guest domains (User Domain (DomU)) through the control interface.
- **User Domain (DomU):** is the guest domain and a VM, which typically has no direct access to physical hardware on the machine. It is often referred to as unprivileged. All para-virtualized VM instances running on a Xen hypervisor must be ported to be Xen-based distributions. Sometimes allowing the User Domain (DomU) guests to have direct access to hardware is required (e.g. a sound card or graphic card).

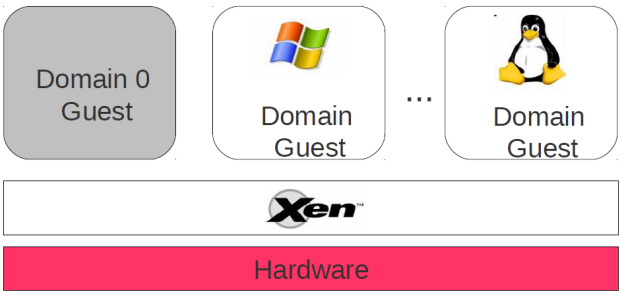

Figure 2.7: Xen-based system architecture [31]

2.4 Cloud Computing

What does the cloud computing term actually mean? So far, this question is still a matter of discussion among researchers. These discussions are nearly coming to an end. The US

National Institute of Standards and Technology (NIST) [11] has defined cloud computing as: "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [70].

The provided definition by Vaquero [72] is, clouds are a collection of usable and accessible virtualized resources. These resources can be reconfigured in a dynamic manner to fit different loads (scales). This would optimize the resource utilization. The pay-per-use model is typically exploited in order to offer guarantees by the infrastructure provider as customized Service Level Agreements (SLAs).

Sriram and Hosseini [70] definition describes the main characteristics of all cloud systems. These features are presented in the following:

- **Virtualization:** the virtualization is an abstraction of the different resources, such as computing and storage of the cloud. The end-user does not realize, where the physical machines and the different types of hardware are located. The Virtualization is intended to disconnect the processing of users in a way that they do not affect each other. Virtualization is a prerequisite for the scalability which is explained below.

- **Scalability:** the scalability represents how the cloud corresponds to changes in the size of a user's needs. In case of larger user demands, additional virtual machines can be started. Scalability is a very important issue in cloud computing, since it enables users to use all resources that they need, which leads to a higher performance at peak demands. Due to the payment nature of cloud applications (i.e., pay per use), there is no loss due to idle time.

- **Accessibility:** resources for cloud applications can be accessed via the Internet. This means, they are available at every time and from everywhere in the world. Accessibility improves the resource sharing. However, the Internet connection may influence the usage of these resources.

- **Quality of Service:** the QoS is a feature offered by most cloud application providers. Negotiations between customers and providers or brokers can take place, and are specified via the SLAs. For example, this specifies guaranteed levels of up-time and penalties against the provider in case of non-compliance with the agreement.

An overview of the deployment and service models of cloud computing systems are presented in Figure 2.8, where three service models could be deployed on top of any of four deployment models. These service models and deployment models are described in the following.

Each cloud provides users with services at a different level of abstraction. In the following the abstraction levels are represented by models known as service models.

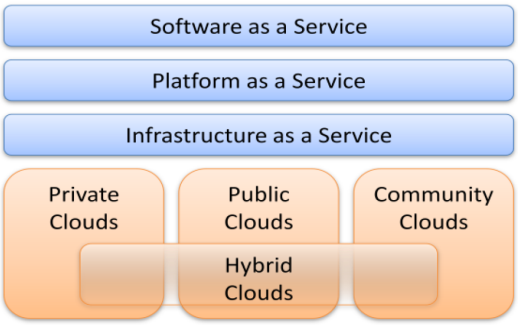

Figure 2.8: Cloud computing deployment and service models [70]

- **SaaS:** at the SaaS model, by using a web-browser, users can access software that others offer as a service over the web. In addition, users do not have the right to control or access the underlying infrastructure, where the software is running [72, 70]. SaaS enables accessing software and data from everywhere without installing software. Therefore, users do not have to care about software updates. Salesforce's [18] Customer Relationship Management software and Google Docs [7] are popular SaaS examples.
- **PaaS:** the PaaS model delivers a cloud platform, which offers a comprehensive environment for users to be able to design, test, and deploy applications. In addition, PaaS hides different types of complexities from the users, such as database access. However, like the SaaS model, users are not allowed to control or access to the underlying infrastructure, which hosts their applications [72]. These applications might be running in the cloud, or in a traditional enterprise data center. The services offered by PaaS are typically virtualized, in order to achieve the scalability required within a cloud. Examples for PaaS model are Microsoft Azure [30], and Google App Engine [6].
- **IaaS:** at the IaaS model, IaaS providers deliver computing resources (e.g. processing power, memory and storage) to users, where they can use these resources to deploy and run their applications. Comparing with the other service models, IaaS model allows users to access the underlying infrastructure through the use of VMs [72, 70]. In addition, users are responsible for updating and upgrading of the OS. Small companies can take an advantage of services based on this model, where they do not have to care about hardware management and its costs (i.e., buying new hardware). Amazon Elastic Compute Cloud (Amazon EC2) [1], Nimbus [10], and Eucalyptus [5] are popular IaaS examples.

Cloud computing deployment models [70] can be classified as private, public, hybrid,

and community clouds. The **private cloud** offers systems and services that are used exclusively by one organization. In contrast, the **public cloud** offers systems and services that are available to the general public usage and can be free or charged. Public clouds are usually owned by large corporations such as Microsoft, Google [6] or Amazon [1]. The **hybrid cloud** is a combination of public and private cloud. In this case, the organization that owns a private cloud can access a public cloud to gain benefits from the services of external providers. The last deployment model is the **community cloud**, which is used and managed by a group of organizations that have shared interests.

Cloud technologies face some of the burden associated with the lack of standardization, where customers are obliged to use cloud providers' proprietary interfaces. Furthermore, security and privacy issues are also arguable, since customers do not know, on which physical machines their data are stored and where these physical machines are located. In case of large amounts of data the reliance on Internet connections results in performance issues. In addition, data and applications can not be accessed if the Internet connection fails [75].

2.4.1 Elastic Utility Computing Architecture Linking Your Programs to Useful Systems (Eucalyptus)

Different custom-made interfaces are provided by cloud system vendors. And the lack of research tools motivates building and developing a common and flexible cloud computing framework to formulate experiments and to address open questions in cloud computing [58]. Eucalyptus [5] is a research-oriented open-source cloud computing system that utilizes resources that are available. It implements IaaS model to provide a solid foundation of cloud systems can be developed and extended. Furthermore, Eucalyptus enables users to create and control VM instances deployed across different physical resources within cloud [57].

The main characteristics of Eucalyptus framework are described in the following:

- The implementation of Eucalyptus components uses Web services technologies, where each component is a stand-alone Web service [57].
- Eucalyptus interfaces are compatible with Amazon EC2 [1]. Amazon EC2 is about to be the origin of cloud computing systems at the market and implements IaaS. Eucalyptus developers want to make it possible to test its functionality and performance against one of the commercial examples of cloud computing systems [58].
- Eucalyptus system supports Kernel-based Virtual Machine (KVM) [9] and Xen [31] virtualization technologies. Eucalyptus components instruct Xen hypervisor to run VMs [57].

- Eucalyptus is designed from the beginning to be easy to install and to be highly modularizable. In this way, researchers can modify existing components (modules) or replace them with their own to address challenging topics in cloud computing systems (VM instance scheduling, construction of virtual networks and definition and execution of SLAs) [57].

2.4.1.1 Eucalyptus Components

An Eucalyptus system [5] has four main components as depicted in Figure 2.9: Node Controller (NC), Cluster Controller (CC), Walrus Storage Controller (Walrus SC), and Cloud Controller (CLC), each has its own WSDL interface. These components are described in the following:

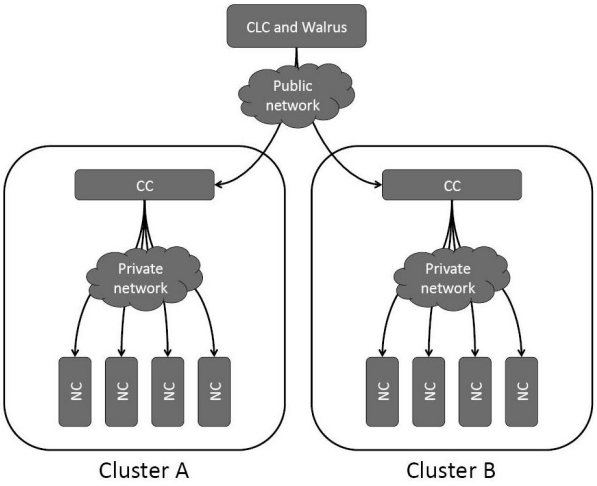

Figure 2.9: Eucalyptus system components [58]

The **Node Controller** is responsible for the creation and the termination of VMs on a physical host. Each machine runs a NC, should have a hypervisor as well. After receiving requests to run or to terminate a VM from the CC, the NC controls the hypervisor, which installed on the same machine, to run or terminate the VM [58].

The **Cluster Controller** runs on a machine that has network connectivity with the machine running the CLC and the nodes running NCs. The CC has three main jobs: the first job is scheduling the requests to run instances on specific NCs. The second one is management of the instance virtual network endpoints as described in Section 2.4.1.2. In addition, it reports information about its NCs. Each CC has a set of NCs. When it receives a request to start an instance, it contacts all NCs belonging to it through *describeResource* operation

and then sends *runInstances* request to the first NC that has enough free resources [58].

The **Walrus Storage Controller** is a storage service included with Eucalyptus system. The Walrus SC interface is compatible with the Amazon Simple Storage Service (Amazon S3) [2]. Walrus allows users to stream data in/out of the cloud and instances. Furthermore, it enables users to manage Eucalyptus VM images (Kernel, Ramdisk and Root filesystem images) [58].

The **Cloud Controller** administrates the whole cloud. Every Eucalyptus system has only one CLC, which is the only visible entry point for users. The CLC makes decisions in Eucalyptus cloud, since it is in charge of handling requests from users and administrators. Furthermore, it is responsible for the high-level scheduling of VM instances by submitting requests to CCs, as well as processing SLAs and maintaining the system and user's metadata [57]. The CLC itself contains of a collection of services as follows [58]:

- *Resource Services:* manage and monitor VMs instances. They communicate with data services to determine and check the references of user request parameters. For example ssh key pairs' parameter to create a VM instance.
- *Data Services:* handle persistent system and user metadata, such as discovering available resource information (VM images and clusters) and handling abstract parameters (e.g., ssh key pairs, security groups and network definitions).
- *Interface Services:* provide web service (SOAP) and web-based interfaces, which handle authentication and protocol translation. These interfaces offer users with different services, such as signing up for cloud access, downloading the cryptographic credentials, which are required for the programmatic interface. In addition, using these interfaces users can query the system (e.g. disk images). Administrators can also manage user accounts and inspect the available system components. Furthermore, Eucalyptus command line tools (Euca2ools) are Linux based text commands, which let users control and manage cloud operations via a standard terminal interface [57].

These services are not static configuration parameters. For example, a user can change the firewall rules that influence the ingress of traffic. These changes are derived in offline mode or are provided as inputs to a resource allocation request.

2.4.1.2 Networking in Eucalyptus System

The VM instance interconnectivity is one of the challenging points in designing a cloud infrastructure. The main characteristics of VM instance network are the connectivity and the isolation [57].

- *Connectivity:* every virtual machine, which is controlled by the Eucalyptus must have a network connectivity to each other and to the public Internet in someway.
- *Isolation:* this means, that VMs belonging to a single cloud allocation have to be able to communicate, but VMs that belong to separate allocations (i.e., users) have to be isolated.

In Eucalyptus system, the CC component is responsible for the set up and tear down of VM instance network interfaces in three different modes. These modes are depicted in the following [57]:

- *SYSTEM Mode:* in this mode, Eucalyptus assigns a random Media Access Control (MAC) address to the VM instance before booting. The VM network interface will be attached to a software Ethernet bridge, which is already connected to the real physical machine's network device. This configuration allows VM network Dynamic Host Control Protocol (DHCP) requests to be performed in the same way as non-VM (non-Eucalyptus) DHCP requests. Then, the VM instances obtain Internet Protocol (IP) addresses.
- *STATIC Mode:* this configuration allows the Eucalyptus administrator to define a static list of MAC address/IP address pairs. Thus, each VM instance at booting obtains a free MAC/IP pair. The allocation of this pair is in first come, first served manner. For each instantiated VM instance, Eucalyptus will maintain a DHCP server with a static entry. Afterwards, the VM network interface will be attached to the real physical network device on the node via the software Ethernet bridge. This mode is useful for assigning of the same list of MAC/IP pairs to the virtualized resources (VMs).
- *MANAGED Mode:* This mode enables the Eucalyptus administrator of fully controlling, managing of VM networks and users of defining of ingress rules (configurable firewalls) for their VM instances. In agreement with the network administrator, the Eucalyptus administrator should specify an IP subnet range that is free for Eucalyptus system to use. Typically this subnet range is a private IP range. Eucalyptus users can define a named network or a security group. Then, they can associate network ingress rules (e.g. Secure Shell (SSH) connection) which apply to any VM running within that named network. Furthermore, the MANAGED mode allows the Eucalyptus administrator of specifying a pool of public IP addresses that users may allocate and assign to their VM instances either at boot or dynamically at run-time.

2.4.2 Comparison between Grid Computing and Cloud Computing

In grid systems, the heterogeneity within the environment (e.g. hardware and software) increases the complexity involved in programmatic and management perspectives. The

development and deployment of grid applications is still difficult and time consuming. Moreover, by using grid middleware applications and libraries, a level of homogenization is applied on the grid fabric layer, i.e., grid resources. As a result of this, resource management inside a particular VO becomes simple. However, developing of grid applications that can be handled by different VOs is still difficult. For example, the co-scheduling of resources across more than one VO is not supported.

In addition, grid interfaces reveal too many details that increase the complexity and time of development. Typically grid interfaces represent the inner capabilities, specially in the case of general purpose grids. Different users perform different usage modes. Therefore, the interfaces should present the complete set of functionalities provided by grid infrastructure. Furthermore, these detailed interfaces hinder the evolution of the middleware and its applications, where many configurations require coding or manual intervention. In addition, users require some level of expertise.

Developing the data and information layers and their services needs a higher abstraction level of the underlying resources in order to automate some actions and interactions that take place inside VOs. Conventional grid systems face some challenges associated with two factors, the resource security settings and the user access privileges (e.g. application deployment on resources).

Wang [75] provides a comparison between grid and cloud computing paradigms. This comparison is summarized in Table 2.1.

	Grid	**Cloud**
Definition	Targets the sharing of distributed computing resources for remote job execution in solving large scale problems.	Offers the users a centric functionality and services to enable building customized computation environments. Cloud targets industry and follows an application driven model.
Infrastructure	Decentralized system distributed in a wide geographical areas. The heterogeneity of hardware and software resources.	Single computing server with a single access point. Centrally controlled despite its spanning several computing centers.
Middleware	Grid has different middleware, such as Globus Toolkit, gLite, and Unicore. These middleware provide different functionalities for resource management, security control and monitoring and discovery.	Cloud has no certain middleware. This is due to the lack of standards. Different issues are still open and unresolved, such as distributed virtual machines management, and distributed storage management.
Accessibility	Grid computing seeks to offer consistent and inexpensive access to large number of reliable distributed resources. However, it is not easy for users to gridify their applications.	Cloud offers customized, scalable, and accessible environments for users.
Application	Grid has several successful applications, such as the LCG project.	Amazon EC2 has gained popularity. However, it is still immature and more applications are necessary to justify the cloud computing paradigm.

Table 2.1: A comparison between grid and cloud computing paradigms

3 Consolidation of Grid and Cloud Systems

Recently, many researches investigate the relationship between grid and cloud computing technologies [50]. Some consider them as competing technologies, where eventually cloud technology will replace grid. Others see that they are complementary. The origin of the confusion about the relationship between grid and cloud technologies is caused by the similarity of their goals, such as minimizing the computing costs and increasing the flexibility, i.e., instead of buying computers using third-party operated hardware.

Grid technology was initiated in the late 1990s. Since that time, it has been highly developed and widely-used. Grid systems enable the dynamic sharing, selection, and aggregation of geographically distributed resources owned by different organizations in order to solve computational and data intensive problems [50].

Although cloud computing technology has appeared few years ago, it has rapidly developed. Cloud technology offers on-demand provisioning of resources such as networks, servers, storage, applications, and services. Virtualization is the factor for enabling cloud computing technology, because it supports the on-demand acquisition of resources and the security by isolation.

Currently, grid systems face some challenges with following scopes: the handling of peak demands, the customization of resources according to users needs, and the burdensome management of heterogeneous resources.

The realization of a hybrid platform combining grid and cloud technologies will provide better services and will be beneficial to resource providers, as well as to end users. On one side, grid systems need to be able to provide dynamically scalable resources and services to end users. This can be accomplished by using cloud technologies for resource provisioning, thus users can customize their environment in order to avoid failure from misconfigured systems or from insufficient allocation of disk space and memory. On the other side, cloud computing systems can benefit from grid computing concepts and services, where they offer federated platforms, geographically distributed computing and storage resources, for distributed and collaborative work. Furthermore, grids provide a set of services, such as such as security, shared data management, system monitoring, and application scheduling and execution. In addition, grid community offers a collection of standard protocols and interfaces, that facilitate interoperability between systems.

3.1 Consolidation Scenarios

The consolidation of cloud and grid systems can be applied in several ways. These ways are classified into four scenarios depending on the use cases of the cloud and grid infrastructure. These scenarios are described in the following:

1. In the **first scenario**, the user of a cloud system can request the services provided by an existing grid system as illustrated in Figure 3.1. AS described in Section 2.2.2, grid systems provide different functionalities for resource management, security control and monitoring and discovery, which can be accessed via specific services implemented within cloud system.

   In some cases, cloud users request resources with specific requirements such as software protected by licenses. These requirements do not exist in the cloud, rather they exist in the grid. Therefore, the cloud needs to extend its capabilities with external resources provided by existing grid systems to resolve such a problem. Another example for this scenario, the cloud user asks for a scheduling service provided by a grid system to manage the execution on VM instances.

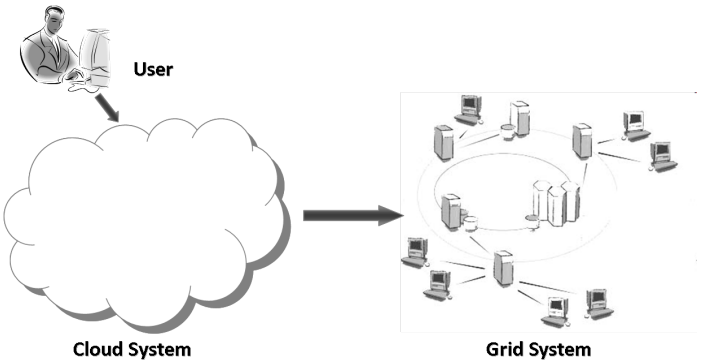

Figure 3.1: The first scenario: the cloud user requests services from a grid system

2. The **second scenario** enables users of a grid system using services offered by a cloud as presented in Figure 3.2. A cloud system provides users with services at a different level of abstraction, i.e., IaaS, PaaS, and SaaS as presented in Section 2.4. For these services, an access point within grid system needs to be implemented.

   One of the implementations of the **second scenario** is concerned with extending the computing capacity in a real grid site with dynamic and scalable resources provided by cloud systems as illustrated in Figure 3.3. In other words, the cloud system provides resources deployed in IaaS environment to the grid. For example, RESERVOIR

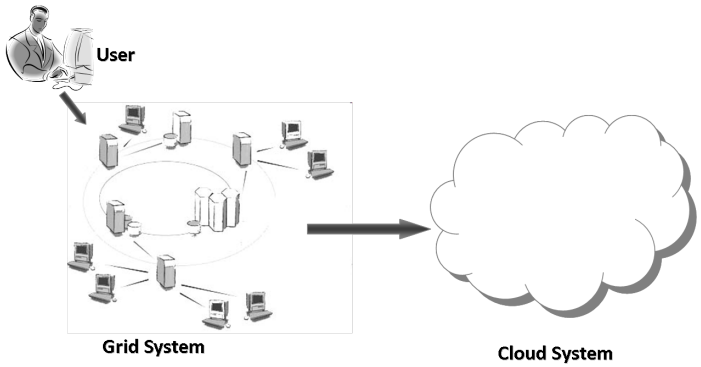

Figure 3.2: The second scenario: the user of a grid system asks for services provided by a cloud system

project [16] supports the on-demand setup and deployment of virtualized resources across administrative domains.

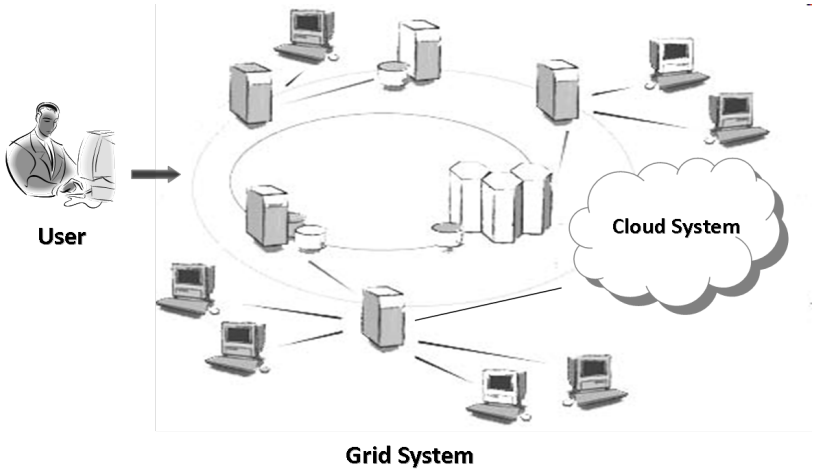

Figure 3.3: Cloud system provides the grid with IaaS model

3. The **third scenario** is a combination of the previous two scenarios, where the user of a system (i.e., grid or cloud) requests services provided by the other one and vice versa as shown in Figure 3.4. This scenario theoretically is possible. However, the implementation of such a system seems to be infeasible, because of difficulty of managing the synchronization of requests from both grid and cloud users. Furthermore, there are many open issues and questions required to be solved and answered to enable the implementation of this scenario. These issues and questions are related to the following domains: open standards, security concerns, resource management,

user management and accounting, which are out of the scope of this thesis.

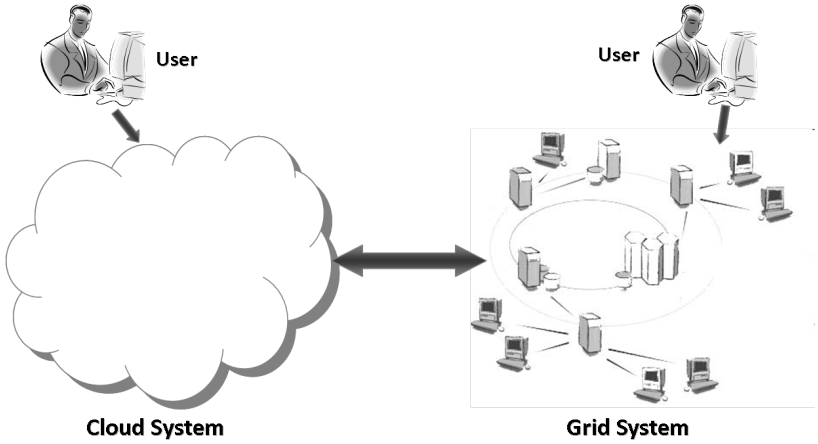

Figure 3.4: The third scenario: users of both grid and cloud systems can request services offered by the other party

4. The **fourth scenario** looks similar to the first scenario. However, this is not really the case, where the cloud user requests a grid system as depicted in Figure 3.5. The provided grid is built on the VM instances offered by the cloud. In this way, we can consider, that a grid system is provided to cloud users as a service, which imposes a Middleware as a Service (MaaS) concept. We proposed the name MaaS, because this service offers a lower level of abstraction than PaaS and a higher level than IaaS. This service, i.e., MaaS virtualizes the whole grid and runs all grid services such as security, monitoring, and scheduling inside the cloud. At IaaS service model the users obtain resources that are delivered to users as a service without a management layer above them, while at PaaS users receive a development environment, i.e., platform as a service to be able to design, test, and deploy their applications.

   This scenario is carried out by the StratusLab collaboration [20], where they conduct an experiment [60] to run a virtual grid inside an Amazon EC2 [1] cloud.

   This scenario increases rapidly the allocation of resources on demand. In addition, cloud systems benefit from grid concepts by federated access control and distributed resource sharing. However, creating a virtual grid site in a cloud raises many questions. The resolve of interoperability problems between virtual grids on different clouds is difficult to implement, because of the lack of cloud computing open standards. In addition, cloud systems are under centralized control, which clearly conflicts with grid system characteristics. Furthermore, cloud systems provide a limited set of features, where the typical cloud storage service, e.g., Amazon S3 [2] is considered as a limited data grid compared to CERN [25] data grid [72].

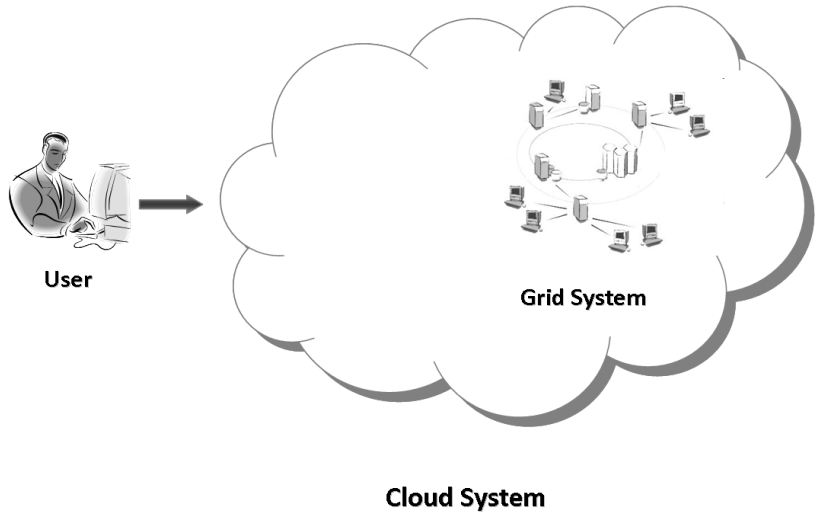

Figure 3.5: The fourth scenario: a cloud system provides users with a grid as service

This thesis adopts the second scenario, i.e., extending grid resources with cloud resources. In the following, benefits that can be gained by employing virtualization and cloud technologies in grid infrastructures are presented [43]:

- **Security:** security classification into system and user level, ensures a high level of security control. At the system level, the host resources consider the VM instances as a complement element. In addition, the VM administrator configures the access privileges inside the VM instance at user level.
- **Performance isolation:** VMs guarantee the performance for both users and applications. Applications that run on VMs do not suffer from the performance perturbation, which is normally invoked by resources competing the concurrent processes on traditional multi-programmed computers.
- **On-demand creation and customization:** VM images can be created and customized by administrators. This results in resource provision for the users. Furthermore, VMs offer reconfigurable infrastructure, which is a concrete advantage for software developers and testers.
- **Legacy system support:** VMs can support legacy environments, i.e., hardware, operating system, and other applications. This can be accomplished by on-demand creation of the relevant VM.

- **Administration privileges:** users of VMs can get the privileges of administrators. Therefore, they can work flexible.
- **Resource Control:** specific VMs are assigned to one user or application. Therefore, it is not necessary to account and control the usage of the resources.

In this Section the combination between grid and cloud has been classified into four scenarios depending on users needs. Section 3.2 illustrates the approaches, that can be applied to realize these scenarios.

3.2 Interoperability Approaches between Grid and Cloud Systems

Interoperability focuses on bringing technologies together in order to benefit from the cooperation features. Wegner [77] defines interoperability term as followed: "Interoperability is the ability of two or more software components to cooperate despite differences in language, interface, and execution platform". Several approaches can be applied to address software components (e.g. of grid and cloud systems) interoperability such as standardization and transformation logic.

Considering a standardization approach, the implementation of open standards does not necessarily mean that the system is interoperable, but rather these standards need to be engineered or improved for interoperability [42]. Standardization approach is a long term solution for interoperability problems. Defining an acceptable standard without prior experience and in-depth study is a difficult mission. In addition, standardization is a slow process, where the standards require time to mature and to get acceptance.

Many organizations support the development of standards for cloud computing such as Open Cloud Consortium (OCC) [14], Open Grid Forum (OGF) OGF Open Cloud Computing Interface Working Group (OCCI) [13], Cloud Security Alliance (CSA) [3], Distributed Management Task Force (DMTF) [4], TeleManagement Forum (TM Forum) [21], and Storage Networking Industry Association (SNIA) [19].

The other transformation logic approach builds an additional component that translates a protocol or schema into another one supported by the receiving component. Adapters and gateways are examples that adopt this concept [64].

An adapter is an extra layer which enables two software components to be connected and translates exchanged information in order to be understandable.

The adapter approach can be implemented either as services within grid and cloud systems or as a stand-alone component. Implementing an adapter as a service within grid and cloud systems means that each system needs to be modified or extended to be able to understand and cooperate the other party. Because of the variety of the custom-made interfaces offered by cloud providers it is not advisable to modify grid systems. Therefore, carrying out the adapter approach within a stand-alone component is the preferable

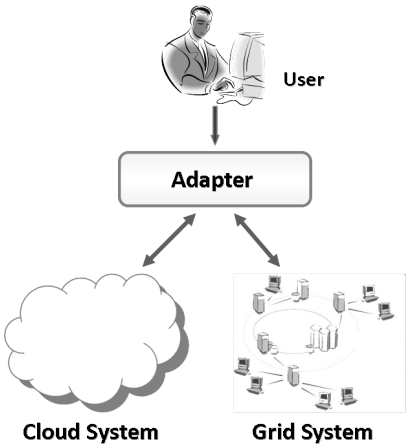

Figure 3.6: Adapter-based approach for grid/cloud systems integration

solution. In Figure 3.6, the user interact with the cloud and the grid through an adapter implemented outside these systems.

A gateway agrees with adapter in terms of concept, but it is implemented as independent entity, that acts as a bridge between different software components.

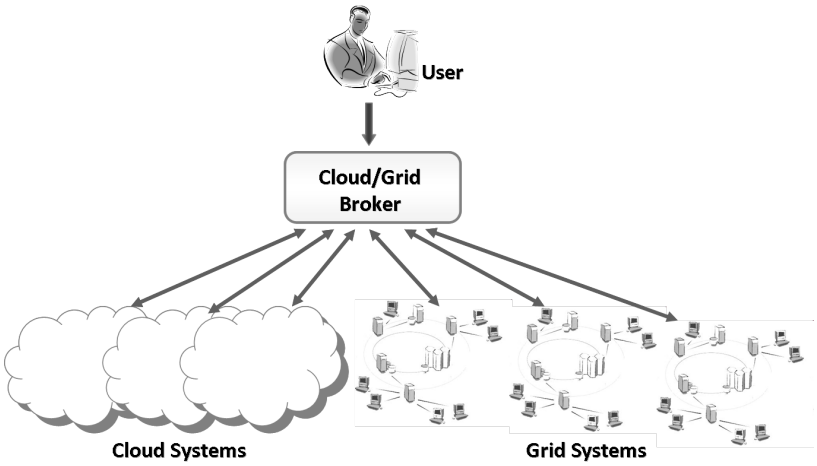

Figure 3.7: Multiple grid/cloud systems integration using a broker

The gateway approach as illustrated in Figure 3.7 named grid/cloud broker. This broker provides a single interface which ca be used by the user to manage and control multiple grids and clouds as well as can benefit from services provided by these grids and clouds.

The main drawback of adapters is that it is a short term solution. A change in the implementation of a component can make the adapter useless. Furthermore, it can be used only by the same components. Since gateways are bridges, they act as a single point of failure or scalability bottleneck.

As described in Section 2.4, cloud systems provide several level of services (SaaS, PaaS, and IaaS). But due to the rapid change and the commercial origination of cloud systems they do not have settled open standard interfaces. Therefore, it is too early to discuss service-level standards considering interoperability. In addition, grid and cloud systems are already built and there is an increasing demand for combining these systems. The market does not have time to wait for standards to appear, but rather, using the other approaches (adapter and gateway) to gain experiences, which later can be used as valuable inputs to the standardization process.

4 A Virtual Runtime Environment for Grid Systems based on Cloud Technology

The conventional execution of grid jobs takes place directly on a cluster of nodes. This way of execution has some limitations. For example, the resources may not be enough to fulfill the requirements of users, operating system of the node may not appropriate or jobs isolation is required. This chapter presents possible design scenarios to allow the execution of grid jobs on virtual clusters. In this way, the above mentioned limitations of the traditional grid systems are addressed. Better job isolation is provided and nodes are customized to fulfill job requirements.

4.1 System Overview

This thesis provides a case study, which focuses on the extension of a grid system by resources provided by a cloud infrastructure. The realization of the case study will follow the adapter interoperability approach described in Section 3.2. The like-adapter solution has to be responsible for handling user requests, i.e., VMs creation and job execution. Furthermore, we will show the successful deployment of the case study by the execution of an application installed on all resources, i.e., grid resources and resources provided by cloud technologies.

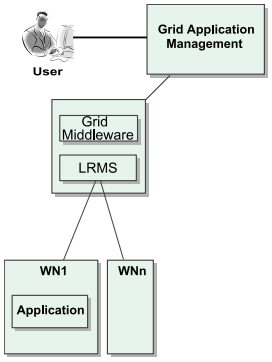

Figure 4.1: The existing system in our infrastructure

A grid system represented in Figure 4.1 is already built in our infrastructure. This system consists of the Grid Application Management (GAM), which carries out the gridification process described in Section 2.2.6. Users submit large tasks to the GAM, where it divides them into smaller jobs. Afterwards, the GAM submits jobs according to their dependencies to grid middleware, which transmits them to the Local Resource Management System (LRMS) to be later executed on Worker Nodes (WNs). Furthermore, the GAM monitors the execution of the jobs, as well as collects and composes the results of these tasks.

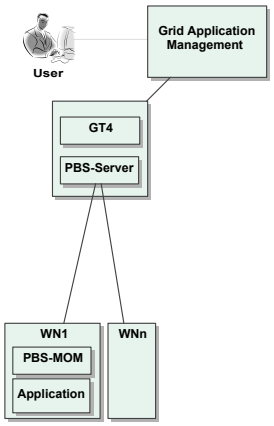

Figure 4.2: The GT4 middleware and the PBS resource manager are technologies installed on the existing system

Figure 4.2 illustrates the deployed technologies on our infrastructure, where GT4 represents the grid middleware, and PBS plays the role of the LRMS.

The drafted schema for our case study is illustrated in Figure 4.3. It consists of two main subsystems a typical grid system and an infrastructure built by cloud and virtualization technologies. In our existing system, the functionality of GAM is extended to enable users to start their VMs and to execute jobs on these VMs.

The GAM should resolve the following issues to perform its functions:

- **Security and accounting:** the communication within the system should be secured and users should be authorized and authenticated to use grid and cloud resources.
- **Data Management:** data storage, access, and transfer have to be enabled, such as data staging in/out for jobs and VM images access.
- **Information Services:** discovery and monitoring of resources should be available.
- **Execution Management:** the deployment, scheduling, and monitoring program execution i.e., jobs and VMs must be handled.

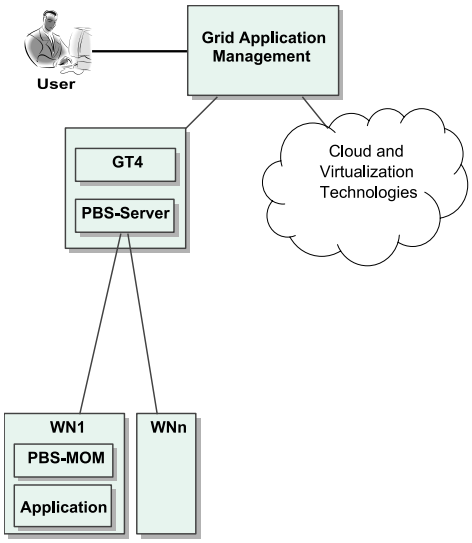

Figure 4.3: The system schema consists of virtualized and non-virtualized subsystems

4.2 System Specifications

In this thesis, several technologies are specified to achieve system goals, i.e., extending grid resources with resources provided by cloud technologies. As mentioned in Section 4.1, our infrastructure has a subsystem which already exists and is based on GT4 as a grid middleware and Tera-scale Open Source Resource and Queue Manager (Torque) PBS as LRMS. Therefore, GT4 plays the role of grid middleware in our infrastructure. In addition, Torque resource manager represents the LRMS, which is responsible for managing (virtualized and non-virtualized) resources and handling job execution in our system.

The extension of resources is performed via cloud technologies, such as Nimbus [10], Eucalyptus, and OpenNebula [15]. All of them are open source cloud computing technologies. Nimbus and Eucalyptus are cloud management toolkits that provide remote and secure interfaces for creating, controlling, and monitoring virtualized resources on their clouds, while OpenNebula is a Virtual Infrastructure Manager (VIM) that provides features to schedule and manage VM instances across multiple physical resources and lacks remote interfaces as the ones provided by cloud toolkits. In other words, cloud toolkits can use VIMs, such as OpenNebula and VMware vSphere [29] instead of managing VM instances directly and can benefit from VIM features [69]. Nimbus toolkit is known as a virtual workspace service, which is developed by globus team and works in conjunction with GT4 services, such as Protocol Extensions to FTP for the Grid (GridFTP) service, which is used by Nimbus storage service [53].

Eucalyptus cloud does not depend on another toolkit as Nimbus and it has remote interfaces, which enable users to interact with the cloud on the contrary of OpenNebula. Therefore, Eucalyptus framework is used to provide our system with virtualized on-demand resources. Eucalyptus was originally developed for research purposes. In addition, its interfaces are compatible with Amazon interfaces to enable the test of performance against one of the popular commercial cloud systems. Xen virtualization technology provides our system with the virtualization layer required by the cloud infrastructure Eucalyptus, where, until now, Eucalyptus supports Xen and KVM virtualization technologies.

In our infrastructure, there are some restrictions, where the resources are low-budget computers. In addition, network settings are configured to guarantee a high level of security, where, on the network, each public IP address is mapped to a specific MAC address. Furthermore, the allocation of public IP addresses is limited.

The relationships and the interactions between these technologies is described in the rest of this chapter and the next one.

4.3 Design Scenarios

The design of the system was through several amendments until it reached its final form, taking into account the working environment and the used technologies. In this section three design scenarios are described to establish a VRE extension for grid. As mentioned earlier in Section 4.1, the system consists of two main parts: virtualized and non-virtualized subsystems. The non-virtualized one is considered as a fixed system. Therefore, it is not further discussed or analyzed. However, the following analysis focuses on the virtualized environment.

4.3.1 One Head-Node for Virtualized and Non-Virtualized Resources

Figure 4.4 represents the first design scenario, where the user specifies and configures the virtualized environment, such as the number of VMs and the operating system of the each VM. After receiving a request from a user, the GAM forwards it to the Eucalyptus system (CLC component) via the command line tool (euca2ools). The GAM waits until all VMs are booted and the PBS-MOM daemons are configured and started. Afterward, the Fully Qualified Domain Names (FQDNs) of VMs are added in the PBS-nodes file of the PBS-server running within the non-virtual environment. In consequence, the PBS-server can monitor and control VM instances throughout their life-cycle via the PBS-MOM running on them. The GAM gives the user a signal, which indicates that the setup of the virtualized environment is done. Afterwards, the user is able to submit jobs to be executed on the VM instances.

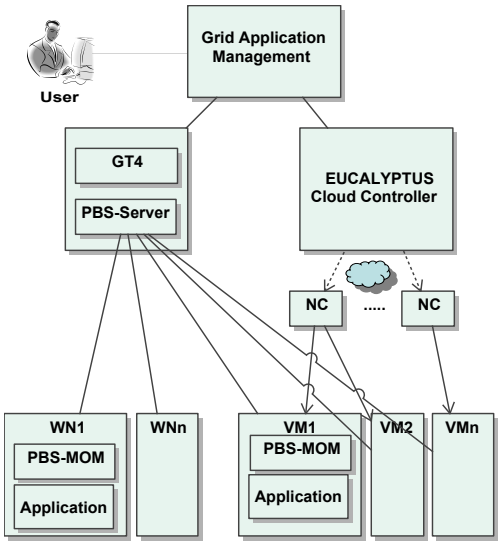

Figure 4.4: The first design: one head-node for virtualized and non-virtualized resources

The implementation of this scenario results in high management effort and costs for maintaining public IP addresses, where each VM instance must have a public IP address to allow PBS-server to manage its PBS-MOM. In addition, on the fixed part (non-virtualized) of the system some modifications should be applied to allow PBS-server to assure, that each user uses only his own VM instances. Therefore, the isolation of jobs execution is difficult to be fulfilled. Furthermore, the modification on PBS-nodes file requires root read-/write privileges to add or remove of FQDNs of VMs. The PBS-server can be considered as a bottleneck in this design, since it handles and manages all jobs execution requests in the whole system (i.e., virtual and non-virtual runtime environments).

4.3.2 A Virtual Cluster for Each User

As shown in Figure 4.5, the user customizes his own virtual cluster including the head node, such as the number of VMs and VM images (head node and worker node VM instance image). Then, the GAM sends a request to Eucalyptus system (CLC component) to start the head node VM instance. When the head node VM instance is booted, the PBS-server daemon and GT4 services (e.g. GridFTP service and globus container) will be started. This is followed by the start of the worker node VM instances including PBS-MOM daemons. The GAM waits until all VM instances run without problems. After that, the FQDNs of worker node VM instances are added to the PBS-nodes file of the PBS-server in the head node VM instance. Afterwards, the PBS-server can monitor and control VM

instances via the PBS-MOM running on them. In addition, the worker node VM instances are informed with PBS-server information, such as IP address and the FQDN. Only the head node VM instance needs a public IP address. After setting up the virtual cluster, the GAM and the user are provided with the public IP address of the virtual cluster's head node in order to enable the use of these virtualized resources.

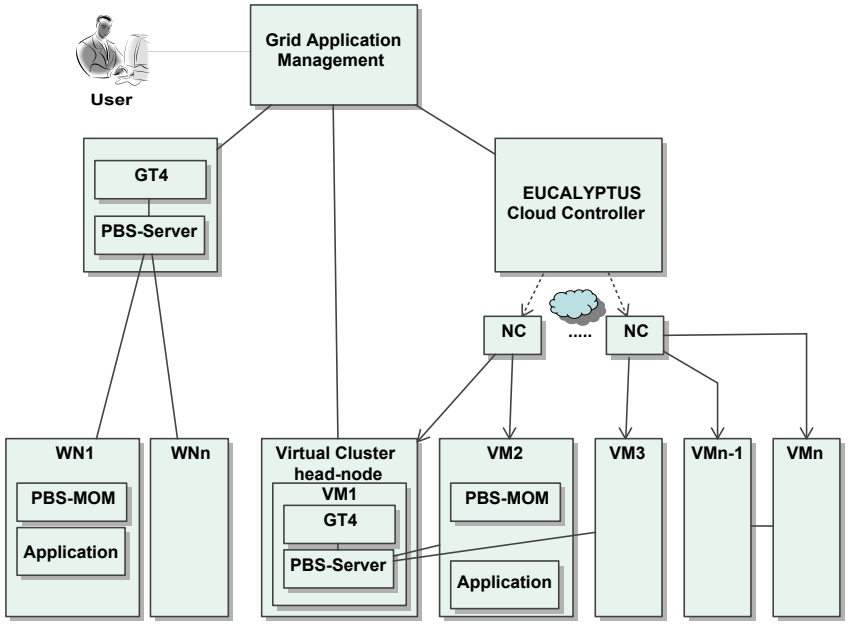

Figure 4.5: The second design: a virtual cluster for each user

The goal of this scenario is to provide a separate virtual cluster owned and managed by the user through the GAM to avoid the bottleneck problem addressed in the first design. Furthermore, this design assures execution environment isolation, where each user has his own cluster. However, after the boot of instances, many configurations have to be carried out. For example, the PBS-MOM daemons must be configured with PBS-server information, as well as the PBS-nodes file should be updated with the new entries (i.e., the FQDNs of VM instances). Furthermore, some of the GT4 configurations can not be automatically derived and human intervention is still needed. For example, certificate requests must be signed by a trusted party (an existing Certificate Authority (CA)) in our site. The required certificate is a host certificate for the head node VM instance, where many GT4 services rely on the existence of this certificate to verify that a host is truly, who it claims to be.

To avoid human intervention and non-automatic configurations, the design can be mod-

ified by placing the GT4 in a non-virtualized resource instead of the head node VM instance. This non-virtualized resource must be within the same network of VM instances. Then, the user or the GAM interact with this non-virtualized resource, where the GT4 exists. This resource hasto be able to communicate with every new PBS-server that starts on a VM instance. Therefore, any of VM instances do not need to allocate a public IP address, since this non-virtualized resource , where GT4 is installed, have a public IP address.

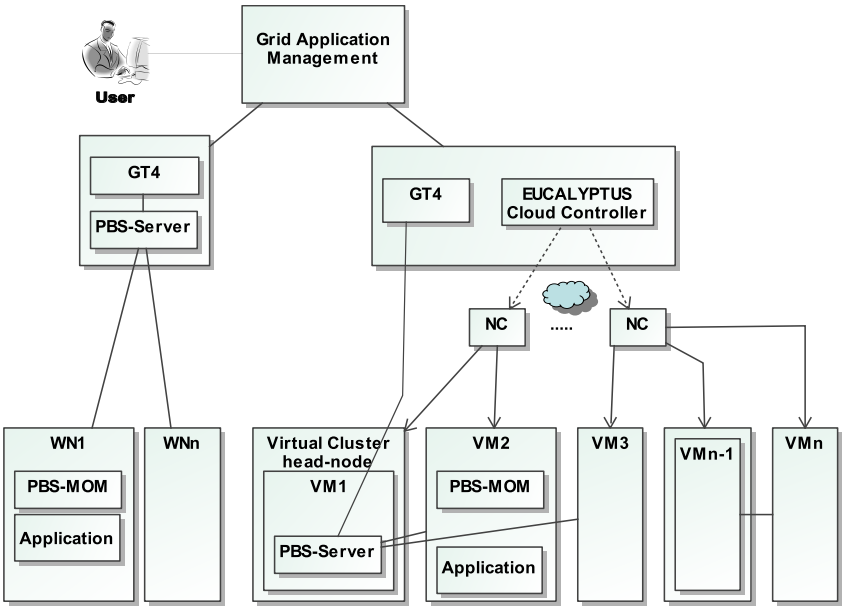

Figure 4.6: The modified second design: a virtual cluster for each user

4.3.3 A User Specific Cluster Queue

The third design scenario is illustrated in Figure 4.7, where two static PBS head nodes are installed. One of them manages the virtualized resources, while the other one manages the non-virtualized resources. The user sends a request to the GAM to create a cluster of virtualized nodes. Therefore, some information, such as the number of VM instances, the VM instance image, and user cryptographic credentials, have to be provided.

The GAM redirects the request to the Eucalyptus component, i.e., CLC to start the VMs and asks the PBS-server to add a queue for this user. This means, the PBS-server manages all VM instances. After the start-up of all VM instances, the FQDN of the VMs will be added to the PBS-nodes file on the machine, where PBS-server runs.

However, more specific configurations in comparison with the two previous design sce-

narios (Sections 4.3.1 and 4.3.2) are required, since queues of users are mapped to their VM instances. At this point, users can submit jobs only to their queues. Each queue has a unique hash coded name, which is not predictable. Therefore, the usage of queues are restricted by their owners. Figure 4.7 represents the mapping of the VM instances to different users (queues) distinguished by circles and stars.

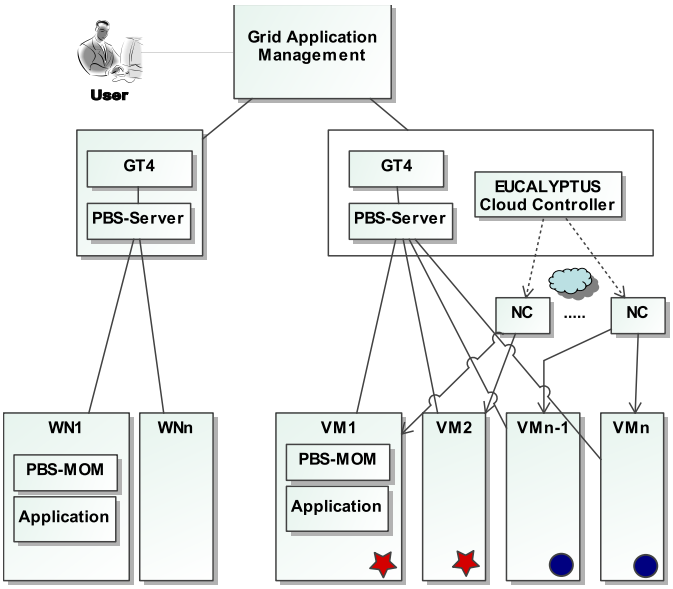

Figure 4.7: The third design: a user specific cluster queue

In this design, as a result of using the queuing method, a satisfactory level of job isolation can be accomplished. However, users may not be able to access their VM instances directly. Furthermore, the sharing of the static cluster head node with other users may result in a bottleneck problem.

4.4 Design Decisions

In order to select one of the proposed design scenarios for implementing the case study, we need to take a close look on system specifications and restrictions placed on our resources and network configurations.

General properties of the test-bed are: the resources are low-budget computers and the network administration requires specific restrictions to increase security level on the network. For example, the allocation of a public IP address is limited. Furthermore, it is

carried by the network administrator and requiring a static and specific MAC address. At this step, the analysis of design scenarios in terms of Eucalyptus networking modes, described in Section 2.4.1.2, is very important, since it validates the appropriate design.

At the first design scenario, each VM instance must have a public IP address. According to the test-bed characteristics , the networking mode within the Eucalyptus cloud must be set to the STATIC mode. In agreement with the network administrator, the cloud administrator defines manually a static list of MAC addresses/public IP addresses mappings, which are allocated to VMs at boot time in a first come, first served manner.

The implementation of the first design is infeasible to integrate in our existing infrastructure for the following reasons: There is no strategy that restricts the use of resources by their owners only. In addition, the costs of allocation of public IP addresses for all VMs is high. Furthermore, root privileges on the head node of the non-virtual enviroment are required to add the FQDNs of VMs in the PBS-nodes file.

The second design scenario provides a better isolation of jobs. The jobs of users can only run on their VMs, thus the data and files of these jobs can not be modified or deleted by other users. Furthermore, this design offer a system that does not suffer from bottleneck problems. However, the implementation is facing some issues due to the strict security settings in our infrastructure network. The allocation of MAC address/IP address pairs is done manually. In this design scenario, only the head node VM instance requires a public IP address.

One of the advantages of this design scenario is the fact that there is no need for allocation of public IP addresses to all VM instances excluding the head node VM instances. In other words, there is no need to reserve a large set of public IP addresses for the Eucalyptus cloud. Therefore, applying of Eucalyptus STATIC mode to allocate IP addresses is infeasible. One of the solutions for allocation of public IP addresses provided by Eucalyptus is the MANAGED mode, which is described in Section 2.4.1.2. In MANAGED mode the administrator can specify a pool of public IP addresses. Users can dynamically at run-time allocate and assign them to their VM instances besides the private IP addresses. However, this mode does not attach the VM instances with MAC addresses. Therefore, we can not allocate public IPs dynamically, which makes it difficult to implement the second design.

Some of the proposed methods for allocation of the public IP addresses from outside Eucalyptus components control are: a) Network Address Translation (NAT) facilities (i.e. Destination NAT (DNAT) and Source NAT (SNAT)), and b) replacing or changing the MAC address for a running VM instance. The new MAC address is supposed to be mapped to a public IP address. However, these methods lead to the unreliability of the VM instance information, since the MAC addresses could be changed without informing the Eucalyptus cloud.

Figure 4.6 represents one change on the second design in order to minimize the required configurations after the boot of VM instances. Although this design does not require allo-

cation of public IP addresses to VM instances, each installation of GT4 can communicate with one and only one PBS resource manager. Therefore, this design can't be implemented.

The most convenient and adequate design scenario is the third one, i.e., A User Specific Cluster Queue, it does not require the allocation of public IP addresses, since the private IP addresses are considered sufficient for implementing the case study. This means that GT4 users are capable of submitting jobs to their VM instances through PBS server. The implementation of this design is presented and discussed in detail in Chapter 5.

5 Implementation

This chapter introduces the realization of the selected design scenario for achieving the interoperability between grid and cloud. It presents the execution of grid jobs on a virtual runtime environment.

This chapter is organized in the following structure. Section 5.1 presents an overview of our system. Afterwards, the test-bed description is provided in Section 5.2. Then, the environment setup and configuration are depicted. The last section shows the concrete execution management on a VRE.

5.1 Overview of System

In order to use the VMs as resources for grid infrastructure, the system provides users with the following functions:

- users shall be able to create, configure, manage and destroy VM instances.
- users shall be able to use these VM instances and execute jobs on their own virtual clusters.

The implementation of a grid virtualized environment needs to cover the following domain areas:

- **Accounting and Security:** In our system, the same user can submit jobs to traditional grid resources and the virtualized ones as well. Therefore, the user needs to obtain a certificate from the grid and user credentials for using the cloud. The user of GT4 must be authenticated by a pair of public and private keys. Firstly, the user has to get a user certificate from a CA, which is installed with GT4. After that, the authenticated user must be authorized for the servers of the grid. The authorization is achieved by adding one line for each grid user in grid-mapfiles that are present on every machine. This line states the certificate in double quotes followed by the UNIX user name as shown in Listing 5.1. These grid-mapfiles map grid certificates to UNIX user names.

Listing 5.1: Registering of a grid user certificate in grid-mapfiles

```
1 "/O=Grid/OU=simpleCA-eucalyptus/OU=ifi.loc/CN=eucalyptus" eucalyptus
```

GT4 increases the security level, where a user accesses the server with his proxy credentials. To prevent user's private key from sending out its secure environment on the user's machine, it signs the public key of a new pair of short-lived proxy credentials. Therefore, the server checks the user's proxy certificate for authentication. After that, it searches for the certificate name in the grid-mapfile and maps the user onto the corresponding UNIX user. GSI provides secure communication that enables single-sign-on. This means, that the user can only login at one machine with his proxy credentials. During the validity period of the proxy certificate, the user is able to use the grid resources.

The creation of VMs typically requires Eucalyptus user credentials. To obtain these credentials, the cloud administrator can add a user by using the Eucalyptus web-based interface. Afterwards, the user can login, and download his/her own x509 certificates and query interface credentials. Note that a GT4 user must be a UNIX user. Unlike Eucalyptus accounts (e.g. administrator and users), that have no relation with user accounts in the operating system. In this thesis, the request to start a number of VMs is submitted as a grid job. This means that the Eucalyptus user credentials are staged-in with the job. The RFT service is responsible for the secure transfer of files. The mechanism of creating VM instances is explained in detail in Section 5.4.1.

- **Data Management:** The system should allow the discovery of information about available VM images and clusters. According to the permissions, which are given to user by the Eucalyptus administrator, users may upload and register their own VM images. Only the Eucalyptus administrator can always upload/register kernels or ramdisks. In Section 5.5, the benefits and drawbacks of allowing the user to upload and register images are discussed. The inquiry about the available VM images and clusters, as well as uploading and registering VM images is carried out by using Eucalyptus query interface (euca2ools command-line tool). However, GT4 users may submit these requests as grid jobs.

- **Information Services:** This service facilitates the discovery and monitoring of VM instances. By using these services, the users can determine the state (e.g. running), the type (e.g. `m1.small`), the IP address, the image `id` of their VM instances. Usually, users can query about their VM instances through Eucalyptus query interface. The execution of the `euca2ools` commands can be performed through a grid job.

- **Execution Management:** This domain is concerned with handling job execution and the creation of VM instances. The GT4 GRAM service is responsible for the jobs execution on VM instances with PBS resource manager. The creation of VM instances takes place through Eucalyptus query interface (euca2ools command-line tool). For the creation of VM instances, this service needs the interaction with other ones, i.e.,

security and data management. We can establish and run VM instances by submitting grid jobs.

5.2 Test-bed

The implementation took place in the Institute of Computer Science at the Georg-August-Universität Göttingen. In our infrastructure the used resources for the implementation are low-budget computers. The properties of these resources are described in Table 5.1.

Resource Properties	**Head Node: ifi01**	**Node: ifi02**
Processor Type	Intel(R) Pentium(R) 4	Intel(R) Pentium(R) 4
Processor Speed	2.4GHz	2.8GHz
Total Memory (RAM)	1005.3MiB	1.7GiB
Operting System	Linux 2.6.31.5-0.1-default i686	Linux 2.6.31.12-0.2-xen i686
Distribution	openSUSE 11.2 (i586)	openSUSE 11.2 (i586)

Table 5.1: A description of system test-bed

5.3 System Setup and Configuration

This section provides a description of the initial steps for the setup and configuration of the grid VRE. The goal of the system is to enable the creation of a private cloud. The physical deployment of the components on resources is represented in Figure 5.1. The head node machine `ifi01` is the entry point for users. Therefore, it should be configured to recieve and perform their requests as described in Scetion 5.3.1. Section 5.3.2 describes the preparing of the node machine `ifi02` with the necessary components, such as Xen hypervisor and the Eucalyptus NC component to enable hosting the VM instances. Lastly in Section 5.4.1 we describe, the preparing of the VM instance image to be able to communicate with Torque server and to execute the jobs.

5.3.1 Head-Node Setup

The head node machine `ifi01` is the entry point to the cloud for all users (e.g. administrators, developer and end-users). Therefore, CLC, CC, and Storage Controller (SC) need to be installed. In case that the design of a system requires more than one cluster in Eucalyptus cloud, the CC and the SC components can be deployed on another machine. Furthermore, the entire GT4 is installed on the head node machine `ifi01`. Torque resource manager will

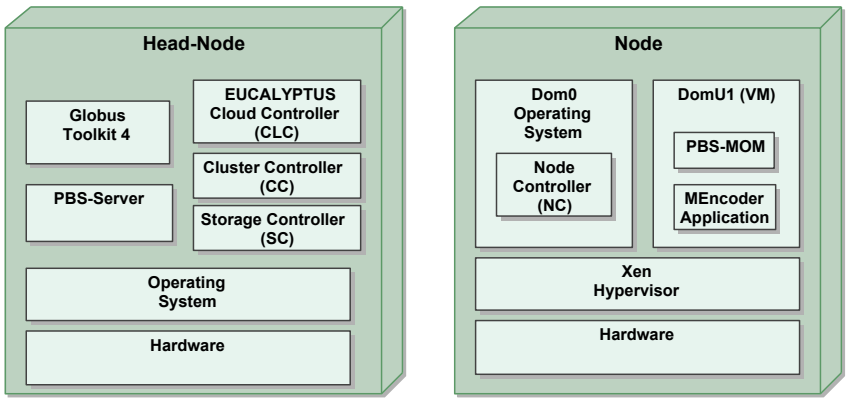

Figure 5.1: The physical overview of the system

be installed and configured to manage the jobs on the virtualized resources. The setup of the head node machine is composed of four main steps, i.e., Eucalyptus installation, network configuration, PBS installation, and GT4 installation. These steps are described in details in the following.

5.3.1.1 Eucalyptus Installation

CLC, CC and SC have been installed from `openSUSE 11` binary packages and according to the web-based Eucalyptus Administrator's Guide, which provides a well defined installation tutorial.

5.3.1.2 Network Configuration

After installing Eucalyptus components on the head node machine Eucalyptus networking configuration needs to be carried out according to the following: The chosen design does not require the allocation of public IP addresses. In addition, in our working environment the allocation of IP addresses typically is restricted by mapping IP addresses to specific MAC addresses. Furthermore, the cost of specifiying a list of IP/MAC mappings, which are provided to VM instances, is high. Therefore, we install an internal DHCP server on the head node `ifi01` to provide the private cloud, i.e., VM instances with private IP addresses. As shown in Figure 5.2 the head node has two Network Interface Cards (NICs). The first NIC named `eth0` allows the public access, i.e., to the Internet and obtains its public IP address from the external DHCP server. Through the second NIC with name `eth1` the nodes (e.g. `ifi02` and the VM instances) contact the internal DHCP server and request IP addresses.

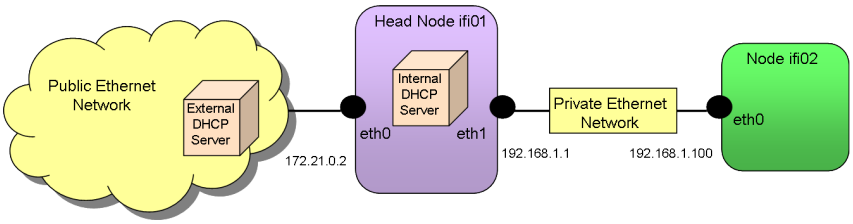

Figure 5.2: The setup of our system network

During the boot-time or run-time the VM instances may need internet access to download or update information and packages. Therefore, the following Listing 5.2 defines a NAT rule to permit private LAN members of public access (Internet) through `eth0` NIC.

Listing 5.2: Enabling private LAN members of public access using NAT rules

```
1 iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

The sufficient networking mode in our case (see Section 2.4.1.2) is the `system` mode, because in this mode the VM instance obtains a random MAC address and a dynamic IP address given by `ifi01` machine's DHCP server. In Eucalyptus's configuration file named `eucalyptus.conf`, the parameter `VNET_MODE` is needed to specify the Eucalyptus networking mode. Furthermore, the variable `VNET_PUBINTERFACE` is set to the device `eth0`, which is connected to the public network. Lastly, the parameter `VNET_PRIVINTERFACE` is set to the device `eth1`, which is attached to the same Ethernet network as the node `ifi01`.

Afterwards, the configuration of the Torque resource manager uses the FQDNs of the VM instances to communicate with the PBS-MOM daemons. Hence, the installation and configuration of Domain Name System (DNS) server on the Head node machine is required and two zones are configured. A DNS zone represents a portion of DNS namespace over which a DNS server has authority. The first zone is the forward zone (e.g. `ifi.loc`), that enables the DNS Server resolving the FQDN to an IP Address. The second is the reverse zone that is used to resolve IP addresses to resource FQDNs on the network. To test the configuration of DNS server `dig` and `host` commands can be used. As shown in Listing 5.3, the `dig` and `host` commands query the FQDN of the host `192.168.1.100`.

Listing 5.3: The query about the FQDN of a host

```
1 # dig +short @192.168.1.1 -x 192.168.1.100
2 cloud100.ifi.loc.
3
4  # host 192.168.1.100
5 100.1.168.192.in-addr.arpa domain name pointer cloud100.ifi.loc.
```

5.3.1.3 PBS Installation

In the third step, the Torque resource manager is installed and configured. However, the PBS nodes file will be empty and dynamically modified. When a VM instance starts, an entry with the FQDN of the VM and a tag, which is associated with a specific queue, is added to PBS nodes file, and is deleted after the termination of the perspective VM. Then, creating self-extracting tarballs, i.e., `torque-package-mom*` and `torque-package-client*` are built via `make packages` command. These tarballs will be installed on computing nodes, i.e. VM instances as illustrated in Section 5.4.1.

In addition, on the head node `ifi01` we export a file system via Network File System (NFS) to the private cloud members, i.e., to the node `ifi02` and to VM instances. This shared file system is used to simplify transferring files and data of jobs within our system.

5.3.1.4 GT4 Installation

In the Last step, a GT4 installation and configuration takes place. This includes the setup of a Simple Certificate Authority (SimpleCA) for signing Globus users certificates, obtaining host certificate for head node machine (`ifi01`), and user certificates. In addition, the globus GRAM service is connected with the Torque server, so that, jobs can be submitted into the PBS batch queue. Then, GridFTP server, Web services Container, and RFT web service need to be configured and started. Afterwards, GT4 users are able to submit jobs, to fork, and to the pre-configured PBS schedulers as well.

5.3.2 Node Setup

The node machine `ifi02` is a physical resource, where Eucalyptus VM instances can run. As mentioned earlier, Eucalyptus and Xen technologies are responsible for starting VM instances in our system. Therefore, Xen hypervisor and Eucalyptus NC component, which communicates with Xen hypervisor to start VMs, are installed. The main steps of the installation and the configuration are described in the following.

- **Xen hypervisor:** It controls the execution of VM instances. The OS of the node `ifi02` is `openSUSE 11.2`. Installing Xen hypervisor in `openSUSE 11.2` is much easier than other OSs (e.g. `Ubuntu`), which has been installed via the graphical user interfaces.

  The Xend daemon [1] file `/etc/xend/xend-config.sxp` is configured as represented in Listing 5.4 to allow all VM instances, i.e., DomUs to appear on the network as individual hosts.

  The `network-script` parameter specifies a script name in `/etc/xen/scripts` directory that will be run to setup the whole networking environment. The `network-bridge`

[1] The Xend daemon is a python application that is considered as a system manager for the Xen environment.

script creates a new bridge named `eth0` and it separates the physical interface of the node from the Dom0 interface. Thus, the firewall setup on Dom0 does not affect the traffic to the VM instances, i.e., DomUs.The `vif-script` parameter specifies a script name in `/etc/xen/scripts` directory that will be run to setup a virtual interface when it is created or destroyed for a VM instance, i.e., DomU. This needs to work in collaboration with the `network-script`.

Listing 5.4: The configuration of Xend daemon

```
1 /etc/xend/xend-config.sxp
2
3 (network-script network-bridge)
4 (vif-script vif-bridge)
```

- Eucalyptus NC interacts with the hypervisor in order to manage the VM instances through `libvirt` [2]. In the NC's configuration file, the parameter `HYPERVISOR` is set to Xen hypervisor. To set up the Eucalyptus network on the node `ifi02`, the primary interface is configured to be a bridge, which is typically set up by Xen hypervisor. The command *brctl show* displays bridge names, which already exist on the machine. In addition, this command lists the VM network interfaces, that are attached to this bridge.

  In Eucalyptus NC's configuration file named `eucalyptus.conf`, the two parameters `VNET_PRIVINTERFACE` and `VNET_PUBINTERFACE` are set to the physical Ethernet device that is attached to the Xen bridge. Furthermore, the parameter `VNET_BRIDGE` is set to the name of the Xen bridge device that is connected to the local ethernet. The bridge is connected to an Ethernet network, that has a DHCP server. The DHCP server is already installed on the head node `ifi01` and is configured to allocate IP addresses dynamically. At this point, the VM instance sends DHCP requests to the local Ethernet, which is replied by the DHCP server on the head node `ifi01`.

5.3.3 Virtual Machine Image Setup

Since the aim of our work is the creation of VM instances to extend grid resources, we prepare the VM image that enables VM instances to act as the other physical resources in our system. In addition, Xen hypervisor represents the virtualization technology in our system, which is based on paravirtualization approach. Therefore, this VM image is configured and customized to be able to run on Xen hypervisor as well.

Our cloud solution is based on the Eucalyptus framework. Therefore, a VM image need to be prepared, so that it can run within Eucalyptus cloud. Eucalyptus Machine Image (EMI) is a combination of a root disk image, kernel, and ramdisk images and their xml files

[2] *Libvirt* is a C toolkit to interact with the Virtualization capabilities of Linux and it provides a stable C Application Program Interface (API) for different Virtualization technologies (e.g. QEMU, KVM and XEN)

containing metadata about them. These images are added separately to Walrus SC and they are considered as templates for VM instances on cloud. The following steps should be followed to prepare a VM image:

Creation of a VM image: we used Ubuntu's vmbuilder package [8] to create the image. It currently supports KVM, Xen, and VMware hypervisors. By using vmbuilder, we are able to pass command line options to add packages, to specify Ubuntu release, and to add a user. Listing 5.5 represents the creation of aVM image using vmbuilder. Vmbuilder's main command line parameter are the virtualization technology (`xen`) and the targeted distribution (`ubuntu`). Futhermore, the example illustrates some optional parameters as well such as Ubuntu relaese (`jaunty`), machine architecture (`arch i386 means a 32 bit`), a default user and password, required packages (e.g. `openssh-server`, `iptables`, and `mencoder`), the mirror to speed the build of image, list of all repositories required by the packages (`main,universe,multiverse`), and the size of the image (`1024 MB`).

Listing 5.5: Creation of a VM image using vmbuilder package

```
vmbuilder xen ubuntu --suite jaunty --flavour virtual -d /home/eucalyptus/ubuntu/
 --arch i386 --user cloud --pass password
 --addpkg python-boto --addpkg portmap --addpkg iptables --addpkg acpid
 --addpkg python-m2crypto --addpkg nfs-common --addpkg openssh-server
 --addpkg make --addpkg mplayer --addpkg mencoder --addpkg joe
 --mirror http://de.archive.ubuntu.com/ubuntu --components main,universe,multiverse
 --rootsize 1024
```

Customization of the image: The resulting image named `root.img` is copied to the head node `ifi01` and modified to be able to run on Eucalyptus cloud and to communicate with the PBS-server on the head node as followed:

- **Mounting the VM image:** First of all, we create a mount point on the head node machine for the image, then associate a free loop block device to the image (e.g. `root.img`) and finally mount it. The steps are presented in Listing 5.6.

Listing 5.6: Mounting of a VM image

```
mkdir mount-point
losetup /dev/loop5 root.img
mount /dev/loop5 mount-point
```

- **Installation of packages:** On the head node machine (`ifi01`), where PBS-server is installed, we have created self-extracting tarballs, i.e., `torque-mom*` and `torque-client*`. After mounting the image we use `chroot` command to install these packages into it. At this point the PBS-MOM is installed on the image. Afterwards, the PBS-server information on this image is configured. Listing 5.7 depicts the configuration of the PBS-MOM, where two files are provided with PBS-server information, i.e., the FQDN of the host where PBS-server runs.

When a job completes, the PBS-MOM copies the output files into the directory from which the job was submitted. By default, copying of this file is accomplished by a remote copy facility such as the secure copy (scp). However, we configure the PBS-MOM to take advantage of the available shared file system NFS on the head node `ifi01` by specifying the `usecp` directive inside the PBS-MOM configuration file as illustrated in Listing 5.7.

Listing 5.7: Configuration of PBS-MOM

```
cat /var/spool/torque/server_name
ifi01.ifi.loc

cat /var/spool/torque/mom_priv/config
$pbsserver ifi01.ifi.loc  # note: IP address of host running pbs_server
$logevent 255
$restricted ifi01.ifi.loc  # note: IP address of host running pbs_server

$usecp *.ifi.loc:/home/*  /home/*
```

We enable the start of PBS-MOM daemon on VM instances at boot time by adding the path of PBS-MOM daemon at the end of `/etc/rc.local` file, otherwise we need to start PBS-MOM daemon manually.

- **Xen-based configuration:** Eucalyptus community provides pre-packaged VM images (e.g. `Ubuntu9.04`, `CentOS5.3`, `Debian5.0`, and `Fedora10`) that are ready to run in the Eucalyptus framework. Each package contains a VM image (`*.img`), a Xen-compatible kernel/ramdisk pair (`xen/vmlinuz*` and `xen/initrd*`), and kvm-compatible kernel/ramdisk pair (`kvm/vmlinuz*` and `kvm/initrd*`).

  The re-configuration and re-customization of these pre-packaged images to fit our requirements is not a trivial task. We use the Xen-compatible kernel/ramdisk pair provided with `Ubuntu9.04` package offered by Eucalyptus community for the prepared VM image via vmbulider. By using this kernel/ramdisk pair, we assure that our VM instance works properly within Eucalyptus framework, as well as with Xen hypervisor. Furthermore, we need the kernel-modules associated with this kernel/ramdisk pair, so that the VM image will be configured correct. We can obtain these kernel-modules by mounting of the pre-packaged image, then copy them outside the mount point. After that, we copy these kernel-modules `./mount-point/lib/modules/` into our customized image.

- **Unmounting the image:** Once all the packages are installed and the modification take place into the image, we should unmount the image and free the loop block device `/dev/loop5`

VM image management within Eucalyptus: To run VM instances from the customized images, we add the disk image (e.g. `root.img`) and the Xen-compatible kernel/ramdisk pair

to Walrus SC using three Elastic Cloud Computing (EC2) commands. Listing 5.8 illustrates an example for bundling, uploading, and registering of an image (`root.img`). In addition, the image is associated with the registered kernel/ramdisk images using their Ids.

Listing 5.8: Registering VM image

```
euca-bundle-image -i root.img --kernel eki-48681661 --ramdisk eri-ABBA17AC
euca-upload-bundle -b ubuntu -m /tmp/root.img.manifest.xml
euca-register ubuntu/root.img.manifest.xml
```

5.4 Execution Management in a Virtual Runtime Environment

This section provides a detailed description of our system in terms of execution management. The user in our system is considered as a grid and a cloud user at the same time. Section 5.4.1 represents the mechanism of creating a VRE via Eucalyptus cloud. The handling of user requests to run a task on the VRE through GT4 middleware is described in Section 5.4.2.

5.4.1 Creation of the Virtual Runtime Environment

To start a VM instance in an Eucalyptus cloud, we need to specify the VM image `ID` and to whom belongs this instance, i.e., user credentials. However, in our system the user should provide the GAM with VM image `ID` VM type, user credentials, the number of VM instances to run, and a seed. This seed is used to generate a unique hash-coded name for the user specific queue, which guarantees the use of instances only by its owner. This Section is concerned with starting VM instances as well as enabling GT4 middleware of submitting jobs to these instances through the PBS-server. The creation of a VRE consists of a sequence of steps: parameter check, adding a queue, starting VM instances, and Integration of VM instances into PBS. A detailed description of these steps is provided in the following.

Parameters check: First of all, we check that all the parameters have valid values. In an Eucalyptus cloud, each user has a limited number of VMs, that can be run. This number is configured via Eucalyptus web-based interface. In our system, the allowed number of VMs is between 1 and 5. Furthermore, we check the validity of user credentials, as well as the existence of the VM image in Walrus SC using `euca-describe-images` command. Listing 5.9 represents an example for user credentials check using corrupted credentials `.euca/eucarc-old`. In order to carry out the user credentials check, we can use euca2ools other commands such as: `euca-describe-instances, euca-describe-availability-zones`.

Listing 5.9: User Credentials Check

```
ifi01:/home/cloud # euca-describe-images --config .euca/eucarc-old emi-B50F148D
Warning: failed to parse error message from AWS: <unknown>:1:0: syntax error
EC2ResponseError: 403 Forbidden
Failure: 403 Forbidden
```

Furthermore, we verify the provided VM image `ID` as well. The `euca-describe-images` command depicts information about the registered VM image, such as the owner of the image, using its `ID` (e.g. `emi-B50F148D`) as shown in Listing 5.10. If there is no return output of `euca-describe-images` command, this means that, the VM image `ID` is invalid.

Listing 5.10: VM Image Existence Check

```
ifi01:/home/cloud # euca-describe-images --config .euca/eucarc emi-B50F148D
IMAGE   emi-B50F148D    ubuntu/ubuntu.img.manifest.xml  admin   available public
        x86_64  machine     eri-ABBA17AC    eki-48681661
```

Adding a queue: Using the seed provided by the user, the name of his queue is computed via `md5sum` command, which computes and checks 128-bit MD5 [3] hashes. With this name, a new PBS queue can be created. Listing 5.11 represents a set of `qmgr` commands, that creates and configures a queue named `eucafd1594c829`. In addition, the 8th line in the Listing maps the queue `eucafd1594c829` to a subset resources using `resources_default.nednode` queue attribute, setting it to a particular node property `euca`.

Listing 5.11: A User Specific Queue Creation

```
qmgr -c "create queue eucafd1594c829 queue_type=execution"
qmgr -c "set queue eucafd1594c829 started=true"
qmgr -c "set queue eucafd1594c829 enabled=true"
qmgr -c "set queue eucafd1594c829 resources_default.nodes=1"
qmgr -c "set queue eucafd1594c829 resources_default.walltime=3600"

# to map a queue to a subset of resources
qmgr -c "set queue eucafd1594c829 resources_default.nednodes=euca"
```

Starting VM instances: By using the command line tool euca2ools VM instances can be started. The `euca-run-instances` command requires at least the VM image `ID` `emi-XXXXX` and user credentials. In addition, the instance must be launched with the appropriate VM type. Otherwise, it will terminate during boot time because of the `insufficient disk capacity remaining` error, i.e., too small VM type. By default, `m1.small` is the VM type that is used for starting a VM. In Listing 5.12, the `euca-describe-availability-zones verbose` command represents VM types and their properties, such as number of processors CPUs and amount of main memory (RAM). In addition, it presents the local cluster (e.g. `euca-cluster-a`), availability details (e.g., free/max CPUs).

Listing 5.12: VM types and their properties

```
ifi01:/home/cloud # euca-describe-availability-zones verbose
AVAILABILITYZONE        euca-cluster-a  192.168.1.1
AVAILABILITYZONE        |- vm types     free / max   cpu   ram   disk
AVAILABILITYZONE        |- m1.small     0001 / 0002   1    128     3
AVAILABILITYZONE        |- c1.medium    0001 / 0002   1    256     7
AVAILABILITYZONE        |- m1.large     0000 / 0001   2    512    10
AVAILABILITYZONE        |- m1.xlarge    0000 / 0001   2   1024    15
AVAILABILITYZONE        |- c1.xlarge    0000 / 0000   4   2048    20
```

[3] Message-Digest algorithm 5 (MD5) [66] is a widely used cryptographic hash function with a 128-bit hash value.

According to the provided number of VM instances, the `euca-run-instances` command is consecutively launched with VM image `ID`, user credentials, and VM type. In Listing 5.13, the `euca-run-instances` command is invoked to run an instance of type `c1.medium` and using the image `emi-29721256`. It returns the instance `ID i-435907D6` and the status of the instance `pending`. Until now, the instance does not have an IP address.

Listing 5.13: Starting a VM instance using `euca-run-instances` command

```
ifi01:/home/cloud # euca-run-instances --config eucarc -t c1.medium emi-29721256
RESERVATION     r-42160749      admin   admin-default
INSTANCE        i-435907D6      emi-29721256    0.0.0.0 0.0.0.0 pending
2010-08-31T04:43:37.313Z        eki-48681661    eri-ABBA17AC
```

We need to check the state of VM instances (e.g. pending, running, or terminated). By using the `euca-describe-instances` command, we check if all VM instances are running and obtain IP addresses. Listing 5.14 represents all information about the instance `i-435907D6` using `euca-describe-instances` command, such as the status of the instance is `running` with IP address `192.168.1.151` in the zone `euca-cluster-a`.

Listing 5.14: VM instance description using `euca-describe-instances` command

```
ifi01:/home/cloud # euca-describe-instances i-435907D6
RESERVATION     r-42160749      admin   default
INSTANCE        i-435907D6      emi-29721256    192.168.1.151   192.168.1.151
running 0       c1.medium       2010-08-31T04:43:37.313Z
euca-cluster-a  eki-48681661    eri-ABBA17AC
```

Integration of VM instances into PBS: Once all instances run with an allocated IP address, we can add their FQDNs to the nodes file of PBS-server. As shown in Listing 5.15 the command `dig` returns the FQDN of the provided IP address. Then, by using `qmgr` commands, we can add a node `cloud151.ifi.loc.` and its property `euca` dynamically to the PBS. This property `euca` can be used for mapping a queue (e.g. `eucafd1594c829`) to this resource, i.e., VM instance.

Listing 5.15: VM instances integration into PBS

```
ifi01:/home/cloud # dig +short @192.168.1.1 -x 192.168.1.151
cloud151.ifi.loc.

ifi01:/home/cloud # qmgr -c "create node cloud151.ifi.loc."
ifi01:/home/cloud # qmgr -c "set node cloud151.ifi.loc. properties = euca"
```

The `qmgr` commands append the PBS-nodes file with a node and its property as shown in Listing 5.16:

Listing 5.16: Syntax of PBS-nodes file after adding a node

```
cloud151.ifi.loc. euca
```

After performing the previous steps, the virtualized resources are mapped to a PBS queue and the VRE is ready to receive and execute user jobs. Figure 5.3 illustrates two queues, where each queue is mapped to a set of VM instances, so that this queue can only use these instances to execute jobs submitted to it.

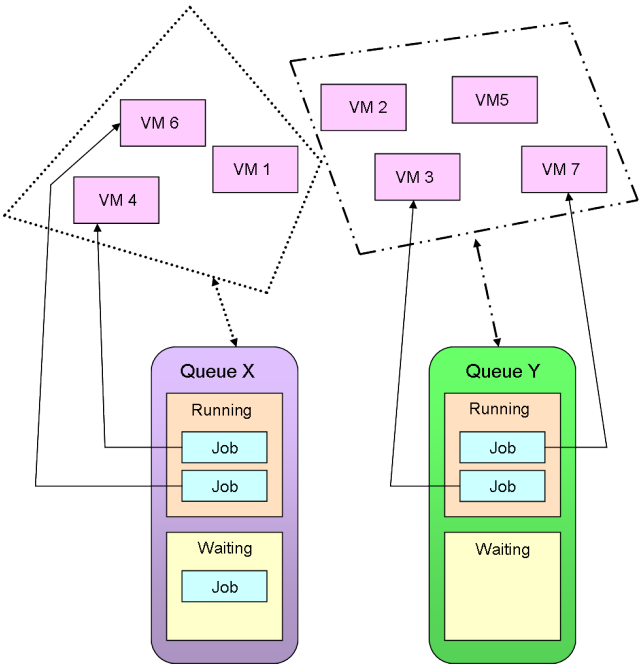

Figure 5.3: Mapping queues to VM instances

5.4.2 Job Execution using the Virtual Runtime Environment

The GAM is responsible for handling tasks submitted by users. According to their requests, the GAM specifies the executing party of submitted jobs, either virtualized or non-virtualized resources. Furthermore, the GAM carries out the gridification process of an application and the parallelization of jobs execution.

The application, which has been installed in our system on the physical resources, as well as on the virtualized ones, is the MEncoder application. MEncoder application is a part of MPlayer suite of tools. MPlayer is a free and open source media player. This application is available for all major operating systems and supports a wide variety of media formats. MEncoder is a close sibling to MPlayer and is a free command line video decoding, encoding and filtering tool.

Listing 5.17 describes how to use the MEncoder command. The MEncoder will slice off the first two minutes of the input movie.

The rest of this section is concerned with integration of MEncoder application into grid as represented in Section 5.4.2.1. After that, Section 5.4.2.2 illustrates the submission of a single job through GT4 client. Afterwards, the submission of a task through GAM is described in Section 5.4.2.3.

Listing 5.17: An example for using MEncoder application

```
torque@grid:/var/local/torque$ mencoder movie.dv -ss 00:00:00
-endpos 00:02:00 -ovc copy -oac pcm -o slice01

where:
 the movie.dv is the input movie
-ss the start position of the slicing
-endpos the end position of the slicing
-ovc Output Video Codec  copy =do not reencode, just copy
compressed frames
-oac Output Audio Codec  pcm = uncompressed PCM audio
-o  the name of output file
```

5.4.2.1 Integration of MEncoder into Grid

The gridification of the MEncoder application is handled by the GAM that communicates with GT4 middleware. GT4 provides the GAM with required services, such as security, data management, and execution management of jobs, to be able to perform its task.

As explained in Section 2.2.6, the application can be dynamically deployed directly before submitting a job or it must be installed beforehand on the resources by administrators. On one hand the dynamic approaches achieves more flexibility in adaptation and a faster employment of new resources. In our system, the MPlayer tools (including MEncoder) has been already installed on all non-virtualized and virtualized resources.

GAM is a script that users can invoke with input parameters (e.g. the input file and the host). It is responsible for splitting large tasks submitted by users into smaller jobs. Afterwards, it determines parameters of smaller jobs and submits them to GT4. In addition, the GAM monitors jobs, and collects and composes the results of jobs.

Figure 5.4 presents a UML sequence diagram that represents a MEncoder gridification process. The user submits a task to the GAM, which splits it into smaller jobs. After that, the submission of jobs to GT4 is done by invoking `globusrun-ws` command. These jobs will be, then, submitted to PBS-server either on the non-virtualized cluster or on the virtualized one through GT4.

After a job has been finished, the output file can be written in the NFS directory by the executing resource. Alternatively, it is possible to use the GT4 service GridFTP instead of the NFS. The GridFTP is a service for staging files between the nodes on the grid. Since the GT4 only has been installed on head nodes of the non-virtualized and virtualized clusters and jobs are transmitted to the PBS resource manager, the PBS is responsible for returning output files.

Once all jobs are completed, the GAM initiates the composition of the results of the jobs. The composed result is returned to the specified directory by user.

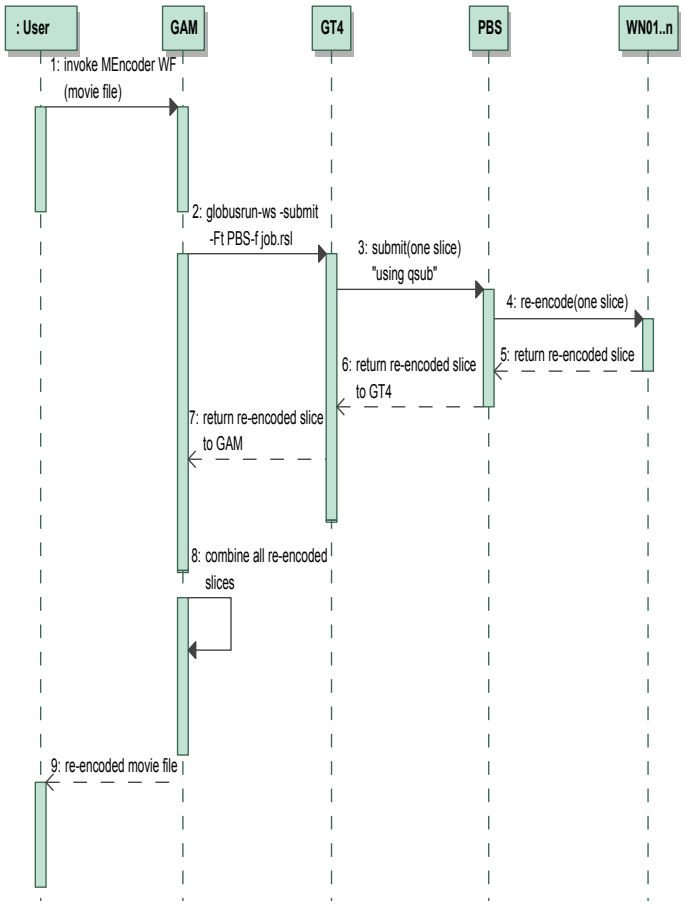

Figure 5.4: MEncoder Gridification Process

5.4.2.2 Submission of a MEncoder Job through GT4 client

The gridification process, which is carried out by the GAM, results in smaller jobs. These jobs need to be described in such a way that enables GT4 to execute them. Therefore, this section concentrates on preparing a template for the job description file of MEncoder jobs. The MEncoder requires input parameters, such as an input file, an output file, an audio encoder, and a video encoder. Therefore, the description of MEncoder jobs to be submitted is provided using the Web Service Grid Resource Allocation and Management (WS-GRAM) eXtensible Markup Language (XML) description syntax.

GT4 offers an XML-based job description language called Resource Specification Language (RSL). RSL offers the ability to include file transfers into the job description. The transfer before and after job execution are called stage-in and stage-out. The RFT service is respon-

sible for staging in/out of files.

Listing 5.18 represents an RSL file of a MEncoder job, that specifies the executable file `mencoder` with some arguments (to achieve the re-encoding of the movie file). In addition, it includes required file transfers and specifies the data transfer rate using rftOpttions tag. It should be noted, that all file location are specified from the viewpoint of the execution host, not of the submission host. In the RSL file we specify a queue (`eucafd1594c829`) within PBS, where our job must be queued. As mentioned earlier in Section 5.4.1, by specifying a queue, the user submits jobs to resources that are already mapped to this queue.

Listing 5.18: An example for MEncoder job description file

```
<job>
        <executable>/usr/bin/mencoder</executable>
        <directory>${GLOBUS_USER_HOME}</directory>

        <argument>slice02</argument>
        <argument>-vf</argument>
        <argument>harddup</argument>
        <argument>-ovc</argument>
        <argument>lavc</argument>
        <argument>-oac</argument>
        <argument>lavc</argument>
        <argument>-lavcopts</argument>
        <argument>vcodec=mpeg4:vqmax=4:acodec=ac3:abitrate=
        128</argument>
        <argument>-of</argument>
        <argument>avi</argument>
        <argument>-o</argument>
        <argument>/home/eucalyptus/torque/mpeg4_slice02</argument>

        <queue>eucafd1594c829</queue>

        <stdout>slice02.stdout</stdout>
        <stderr>slice02.stderr</stderr>

        <fileStageIn>
           <transfer>
             <sourceUrl>gsiftp://ifi01.ifi.loc/var/local/torque/slice02</sourceUrl>
             <destinationUrl>file:///${GLOBUS_USER_HOME}/slice02</destinationUrl>
             <rftOptions>
                 <parallelStreams>4 </parallelStreams>
             </rftOptions>
           </transfer>
        </fileStageIn>
</job>
```

The submission of a MEncoder job through GT4 client using `globusrun-ws -submit` command is presented in Listing 5.19. The job is described in an RSL file and must be transmitted to PBS recourse manager, so that the PBS is responsible for choosing the execution resources.

Listing 5.19: Submission of a MEncoder job through GT4 client

```
eucalyptus@ifi01:~> globusrun-ws -submit -F
            https://ifi01.ifi.loc:1024/wsrf/services/ManagedJobFactoryService
            -Ft PBS -f encode-slice-sample.rsl
```

5.4.2.3 Job Submission through Grid Application Management

GAM has a Python script used to drive a complete workflow. It initially splits up the large movie into N pieces, the length of the piece is L minutes. Each piece is submitted to GT4, where GT4 forwards the job to PBS resource manger to be re-encoded by MEncoder application. The script waits for the jobs to finish and then combines the results together into a newly encoded version of the movie file.

Listing 5.20 shows the invoking of this script `mencoderWorkflow`, where it requires paths of the input and output movies, the name of the host, where the PBS WS-GRAM job manager runs. In addition, the script requires a PBS queue name in case that `virtual` parameter is set to `true`. The `virtual` parameter specifies if jobs are supposed to run on the virtualized or non-virtualized resources. This script is originated from The Globus Consortium [24], which provides a GT4 compute grid tutorial. This script is adapted and modified to fit our system.

Listing 5.20: Invoking `mencoderWorkflow` script

```
eucalyptus@ifi01:~>  ./mecoderWorkflow --input=./aMovie.avi
--output=./re-encoded-aMovie.mp4 --PBShost=ifi01.ifi.loc
--virtual= true   --Queue=eucafd1594c829
```

The goal of `mencoderWorkflow` script is to implement the gridification process, which consists of several steps. These steps are described as follows:

- **Proxy credential verification:** Firstly, we should check if the user has a valid proxy credential using the `grid-proxy-info -e` command.
- **Task splitting into smaller jobs:** the movie file is sliced into N pieces, where each one is of L minutes length. Each slice is a MEncoder job that is described in an RSL file as presented in Section 5.4.2.3. Afterwards, these MEncoder jobs are submitted to GT4 via `globusrun-ws -submit` command. Then, GT4 transmits them to PBS with the provided queue name in case the user want to use the virtualized resources. In our system, queue solution is implemented to assure that users use only their VM instances.
- **Job status checking:** The status of all submitted encoding jobs is queried using `globusrun-ws -status -job-epr-file` command. We keep checking and monitoring their progress until they are all finished. After that, we can stitch back the newly encoded slices.

5.5 Discussion

The consolidation solution, i.e., GAM implemented in this thesis is composed of two parts. The first part is responsible for the creation of the VRE according to user specifications. The second part is concerned with applying the gridification process for the MEncoder application and the execution of user tasks.

The realized system depends on Xen hypervisor to create the VRE. Since Xen implements the paravirtualization approach, the VM image needs to be Xen-based. Therefore, the creation of a VRE requires providing a registered VM image `ID` in Eucalyptus Walrus. In other words, users have to configure and register their VM images in Eucalyptus Walrus manually before running VM instances. In this way the correct uploading and registering a VM image is guaranteed and these images can be tested by the user. However, starting VM instances with an on-the-fly image files is not allowed, and the dynamic customization of VMs must be done by the users themselves.

In our system, the termination of a VRE is accomplished according to user's request. Furthermore, before releasing a VRE, the system checks if each VM instance in this VRE is free and there is no job running on it. The termination of the VRE is done either after a long period or a short period of the start-up time of VM instances, which influences the resources performance in different ways.

For example, using short-lived VM instances several times by users increases the time required for booting and terminating these instances. This will correspondingly increases the waste time from the resource perspective. The exact effect of such a problem needs more investigations. However, it will not be discussed here because it is out of the scope of this thesis.

The part that applies the gridification process in our system is application-dependent. Therefore, when the user wants to use or deploy another application on VM instances, this part has to be adjusted for the new application.

In this thesis, the gridification process of the MEncoder application requires slicing the task (a media file) into smaller jobs. These jobs are submitted to be transformed into another format according to user requests. There is no specific order needed to be followed for the execution of these jobs. However, this is not the case for all applications. For example, some scientific programs are more sophisticated and require additional considerations, such as BLAST, the bioinformatics application, and Montage, the astronomy application. Such applications require additional analysis for the determination of the smaller jobs. This difficulty is resulted from the high dependency and interconnection inside the large task. Furthermore, the order of jobs execution has to be determined.

In our implementation, some burdens have been detected. They are described in the following. The Eucalyptus framework, like other cloud computing technologies, does not provide users with information about physical hosts of their VM instances, rather it provides only the zone or the CC, to which the VM belongs. The only possible way to identify

the location of the VM instance is to search manually in each NC belonging to this CC.

Furthermore, the greedy and round-robin scheduling algorithms are implemented by Eucalyptus framework for the allocation of VM instances on specific physical hosts. These algorithms are seen as simple scheduling policies for such advanced systems, where no load or task information is used when making a scheduling decision.

In addition, the modification on PBS-nodes file and adding a new PBS queue requires root or PBS-administrator privileges, which may raise security concerns.

6 Conclusion

This chapter concludes the thesis by providing a summary for it. Furthermore, an overview of work related to this thesis is briefly depicted. The chapter finally discusses the possible extensions for this work as an outlook.

6.1 Summary and Discussion

Over time, researches get more involvement in designing and building the infrastructure needed to handle computational and data intensive tasks of complex scientific applications. One of the promising solutions is the grid computing concept, which was initiated in 1990s. However, the grid community still faces different challenges like fulfillment of peak demands, customized computing environment provision, and complicated management of heterogeneous resources. This increases the awareness to harness cloud technologies to address grid limitations.

Different consolidation scenarios of grid and cloud systems were discussed in this thesis. In addition, the interoperability approaches (e.g., standardization, adapters, and gateways) that can be applied to bring grid and cloud systems together were described. Since cloud computing standards are still under development, the realization of consolidation scenarios based on the standardization interoperability approach seems to be infeasible. The gateway approach is not considered because our system does not integrate multiple grids and clouds. Based on these facts, the adapter solution is followed to enable the interoperability between cloud and grid systems. Such a solution can be used to gain experiences and to provide valuable inputs for the standardization process.

One of the discussed consolidation scenarios was the extension of a grid system by cloud services scenario. This scenario was validated in a case study in this thesis. Different design scenarios for an extension for a GT4 grid environment with VM instances provided by Eucalyptus framework were presented. The strengths and weaknesses of each scenario were discussed as well. The selected scenario is based on a user specific cluster queue design, which defines a queue associated with a set of VM instances for each user. In this way, the use of VM instances is restricted to their owners only, which provides a better isolation for jobs, i.e., the files and data of the job can not be modified or deleted by another user.

A description for the implementation of the selected design scenario was provided in this thesis. At first, the setup of the working environment including the preparation of the Xen-based VM image was described. After that, the execution of an application on grid resources as well as on a Virtual Runtime Environment provided by Eucalyptus framework was carried out to test of the deployment of the system.

In our system, users are responsible for providing the VM image to start the VRE. In this way, we avoid the mis-configuration of requested resources from cloud-grid infrastructure perspective. At periods of peak workloads, the user benefits from the creation of a VRE, since grid conventional and non-virtualized resources are fully occupied. Moreover, the virtualization layer increases the utilization of the system.

The burdens raised by the implementation in the case study are discussed in the following. The implemented system in this thesis provides users, who create VREs, with names of their PBS-queues, where jobs will be placed. The configuration of these PBS queues and nodes, i.e., the VM instances requires a root or at least PBS-administrator privileges, which results in security concerns.

Furthermore, the Xen hypervisor represents the virtualization layer in our system, which obligates the portion of the VM image to be Xen-based. The build of Xen-based VM images is not a trivial task, because of the lack of documentations and tutorials describing, such as configurations. In addition, it requires the expertise in Linux and Xen configurations. In our work, building a VM image required about three weeks of research and attempts driven. However, the configuration of a VM image needs to be done all at once. Then, this image will be registered in the images repository in the cloud system.

One of the common burdens associated with most of cloud computing infrastructures including the Eucalyptus framework is the inability of obtaining any information about the physical hosts of VM instances. In addition, this system has to be adjusted in order to integrate a new grid or cloud infrastructure.

6.2 Related Work

This Section provides a summary of related work that has common features with different aspects of this thesis.

The integration of virtualization technologies in grid infrastructures has been previously explored by several projects. For example, the In-VIGO project [33] proposed a distributed grid infrastructure based on VMs.

In [75], a light weight middleware, the Grid Virtualization Engine (GVE), is designed and implemented to enable building multiple Virtual Distributed Environments (VDEs) on grid infrastructures working with virtualization technologies, i.e.,VM hypervisors [76].

The Cumulus project focuses on building a scientific cloud presented by Wang [75]. They integrate existing grid infrastructures with cloud technologies, i.e., Globus Virtual

Workspace service (Nimbus) and OpenNebula by building frontend and backend services. The frontend service re-engineers the Globus Virtual Workspace service, while OpenNebula acts as Cumulus backend.

The StratusLab project [20] aims at developing a toolkit that integrates cloud and virtualization technologies and services within grid sites. StratusLab current activities focus on the identification and evaluation of different integration scenarios of cloud and grid infrastructures.

The Enabling Grids for E-sciencE (EGEE) project works with RESERVOIR [16] to support the EGEE sites. So that, they can benefit from adopting cloud models to meet the changing needs of users. The RESERVOIR cloud infrastructure solution is built on OpenNebula project, the open source toolkit for cloud computing [17].

The VGrADS project [27] developed software systems that simplify and accelerate the development of applications to run on distributed systems such as grid and cloud infrastructures [62].

6.3 Outlook

This thesis discussed consolidation scenarios of grid and cloud systems and presented an initial and promising step toward a better integration between them. The realization of such an integration requires further extensions for the work described in this thesis. In the following, an illustration for the possible extensions for this work is provided.
Additional functionalities can be integrated into the solution created in the thesis to act as a grid/cloud broker. It is supposed to support multiple grid and cloud systems and enable users to use services provided by them.

A further mechanism can be added to allow the dynamic and automatic creation of a VRE to handle peak workloads and release when the load decreases. In other words, our work can be modified to be sensitive and intelligent to the workload.

The integration solution can be extended to support the gridification of more applications to accomplish users needs. As a result, the system should allow the dynamic deployment of applications on VM instances and be able to handle large tasks that should be executed by these applications. This can be achieved by integrating workflow frameworks such as Pegasus [26], Chimera [49], and ActiveBPEL engine [22] with our work. Workflow frameworks construct abstract high-level descriptions for applications named workflows and map them onto grid environments. These workflows reflect the structure and the behavior of the applications in a platform independent manner.

List of Abbreviations

ABI Application Binary Interface
Amazon EC2 Amazon Elastic Compute Cloud
Amazon S3 Amazon Simple Storage Service
API Application Program Interface
B2B Business-to-business
CA Certificate Authority
CC Cluster Controller
CLC Cloud Controller
CORBA Common Object Request Broker Architecture
CPU Central Processing Unit
CSA Cloud Security Alliance
CSF Community Scheduler Framework
DHCP Dynamic Host Control Protocol
Dom0 Domain0
DomU User Domain
DMTF Distributed Management Task Force
DNAT Destination NAT
DNS Domain Name System
EC2 Elastic Cloud Computing
EGEE Enabling Grids for E-sciencE
EMI Eucalyptus Machine Image
Eucalyptus Elastic Utility Computing Architecture Linking Your Programs to Useful Systems
FTP File Transfer Protocol
FQDN Fully Qualified Domain Name
GAM Grid Application Management
GGF Global Grid Forum
GRAM Grid Resource Allocation and Management
GridFTP Protocol Extensions to FTP for the Grid
GSI Grid Security Infrastructure
GT4 Globus Toolkit 4
GVE Grid Virtualization Engine
HaaS Hardware as a Service

HTTP Hypertext Transfer Protocol
IaaS Infrastructure as a Service
IP Internet Protocol
JNDI Java naming and directory interface
KVM Kernel-based Virtual Machine
LAN Local Area Network
LHC Large Hadron Collider
LRMS Local Resource Management System
LSF Load Share Facility
MaaS Middleware as a Service
MAC Media Access Control
MD5 Message-Digest algorithm 5
MMJFS Master Managed Job Factory Service
MOM Machine Oriented Miniserver
MP4 stands for MPEG-4 Part 14
MPEG-4 Motion Picture Experts Group Layer-4 Video
NASA National Aeronautics and Space Administration
NAT Network Address Translation
NC Node Controller
NFS Network File System
NIC Network Interface Card
NIST National Institute of Standards and Technology
OCC Open Cloud Consortium
OCCI OGF Open Cloud Computing Interface Working Group
OGF Open Grid Forum
OGSA Open Grid Service Architecture
OGSA-DAI Open Grid Services Architecture Data Access and Integration
OS Operating System
PaaS Platform as a Service
PBS Portable Batch System
PC Personal Computer
POSIX Portable Operating System Interface for UNIX
QoS Quality of Service
RFT Reliable Transfer Service
RMI Remote Method Invocation
RMS Resource Management System
RSL Resource Specification Language
SaaS Software as a Service
SC Storage Controller
SimpleCA Simple Certificate Authority

SLA Service Level Agreement
SNAT Source NAT
SNIA Storage Networking Industry Association
SOAP Simple Object Access Protocol
SSH Secure Shell
SSI Single System Image
TCP/IP Transmission Control Protocol/Internet Protocol
TM Forum TeleManagement Forum
Torque Tera-scale Open Source Resource and Queue Manager
UDDI Universal Description, Discovery, and Integration
UML Unified Modeling Language
VDE Virtual Distributed Environment
VIM Virtual Infrastructure Manager
VO Virtual Organization
VM Virtual Machine
VMM Virtual Machine Monitor
VRE Virtual Runtime Environment
Walrus SC Walrus Storage Controller
WebMDS Monitoring and Discovery Service of Globus Toolkit 4 via the Web Browser
WMS workflow management system
WMV Windows Media Video
WN Worker Node
WS Web Service
WSDD Web service deployment descriptor
WSDL Web service description language
WS-GRAM Web Service Grid Resource Allocation and Management
WSRF Web Services Resource Framework
XML eXtensible Markup Language

Bibliography

[1] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/, Last visited: 16 September, 2010.

[2] Amazon Simple Storage Service. http://aws.amazon.com/s3/, Last visited: 16 September, 2010.

[3] Cloud Security Alliance (CSA). http://www.cloudsecurityalliance.org/, Last visited: 16 September, 2010.

[4] DMTF - Distributed Management Task Force. http://www.dmtf.org/, Last visited: 16 September, 2010.

[5] Eucalyptus Systems. http://open.eucalyptus.com/, Last visited: 16 September, 2010.

[6] Google App Engine. http://code.google.com/intl/de-DE/appengine/, Last visited: 16 September, 2010.

[7] Google Docs - Online documents. http://docs.google.com/, Last visited: 16 September, 2010.

[8] Jeosvmbuilder. https://help.ubuntu.com/community/JeOSVMBuilder, Last visited: 16 September, 2010.

[9] Kernel Based Virtual Machine. http://www.linux-kvm.org/page/Main_Page, Last visited: 16 September, 2010.

[10] Nimbus. http://www.nimbusproject.org/, Last visited: 16 September, 2010.

[11] NIST Cloud Computing. http://csrc.nist.gov/groups/SNS/cloud-computing/, Last visited: 16 September, 2010.

[12] OASIS Web Services Resource Framework (WSRF). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf, Last visited: 15 September 2010.

[13] OGF Open Cloud Computing Interface Working Group. http://occi-wg.org/doku.php, Last visited: 16 September, 2010.

[14] Open Cloud Consortium (OCC). http://opencloudconsortium.org/, Last visited: 16 September, 2010.

[15] OpenNebula. http://www.opennebula.org/, Last visited: 16 September, 2010.

[16] RESERVOIR - Resources and Services Virtualization without Barriers. http://www.reservoir-fp7.eu/, Last visited: 16 September, 2010.

[17] RESERVOIR and Enabling Grids for E-sciencE (EGEE). http://62.149.240.97/index.php?page=egee, Last visited: 15 September 2010.

[18] Salesforce. http://www.salesforce.com/eu/, Last visited: 16 September, 2010.

[19] Storage Networking Industry Association (SNIA). http://www.snia.org/home/, Last visited: 16 September, 2010.

[20] StratusLab. http://www.stratuslab.org/doku.php, Last visited: 16 September, 2010.

[21] TeleManagement Forum (TM Forum). http://www.tmforum.org/browse.aspx, Last visited: 16 September, 2010.

[22] The ActiveBPEL Engine. http://www.activebpel.org/, Last visited: 15 September, 2010.

[23] The Global Grid Forum. http://www.ggf.org/index.php, Last visited: 16 September, 2010.

[24] The Globus Consortium. http://www.globusconsortium.org/index.html, Last visited: 16 September, 2010.

[25] The Large Hadron Collider. http://public.web.cern.ch/public/en/LHC/LHC-en.html, Last visited: 16 September, 2010.

[26] The Pegasus Project. http://pegasus.isi.edu/, Last visited: 15 September 2010.

[27] The VGrADS Project. http://vgrads.rice.edu/, Last visited: 15 September 2010.

[28] VMware. http://www.vmware.com/, Last visited: 16 September, 2010.

[29] VMware vSphere. http://www.vmware.com/products/vsphere/, Last visited: 16 September, 2010.

[30] Windows Azure Platform. http://www.microsoft.com/windowsazure/, Last visited: 16 September, 2010.

[31] Xen Hypervisor. http://www.xen.org/, Last visited: 16 September, 2010.

[32] T. Abels, P. Dhawan, and B. Chandrasekaran. An Overview of Xen Virtualization. Dell Power Solutions, August 2005.

[33] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu. From Virtualized Resources to Virtual Computing Grids: the In-VIGO System. *Future Generation Computer Systems*, 21(6):896--909, 2005.

[34] M. Baker and R. Buyya. Cluster computing at a glance, 1999.

[35] P. Barham, B. Dragovic, K. Fraserr, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles, Bolton Landing, NY, USA*, SOSP '03, pages 164--177, New York, NY, USA, 2003. ACM.

[36] A. Bayucan, R. L. Henderson, J. P. Jones, C. Lesiak, B. Mann, B. Nitzberg, T. Proett, and J. Utley. *Portable Batch System Administrator Guide*. Veridian Systems, Inc., pbs protm 5.1 edition, June 2001.

[37] R. Berlich. Grid Computing: Roots, Motivations and Implementation. http://www.egee.nesc.ac.uk/trgmat/events/040920GridKa/talks/slides/whatIsGrid.pdf, Last visited: 16 September, 2010.

[38] S. Brasol. Analysis Of Advantages And Disadvantages To Server Virtualization. Technical report, Bowie State University, December 2005.

[39] R. Buyya. *High Performance Cluster Computing: Architectures and Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[40] Rajkumar Buyya, Toni Cortes, and Hai Jin. Single system image. *Int. J. High Perform. Comput. Appl.*, 15(2):124--135, 2001.

[41] Platform Computing Corporation. *Platform LSF Administrator's Guide*, June 2001.

[42] L. Field and M. Schulz. Grid Interoperability: the Interoperations Cookbook. *Journal of Physics: Conference Series*, 119(1):012001, 2008.

[43] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes. A Case For Grid Computing On Virtual Machines. In *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 550, Washington, USA, 2003. IEEE Computer Society.

[44] I. Foster. Globus Toolkit Version 4: Software for Service Oriented Systems. In *In Proceedings of the IFIP International Conference on Network and Parallel Computing (NPC05)*, volume 3779 of LNCS. Springer, 2005.

[45] I. Foster. *What is Grid? A Three Point Checklist*. GridToday, July 20, 2002.

[46] I. Foster, Fujitsu H. Kishimoto, and Fujitsu A. Savva. The Open Grid Services Architecture, Version 1.5. `http://www.gridforum.org/documents/GFD.80.pdf`, Last visited: 16 September, 2010, July 2006.

[47] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure*, chapter 2. Morgan Kaufmann Publishers, August 1998.

[48] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal Of Supercomputer Applications*, 15(3):200--222, 2001.

[49] I. Foster, J. Vöckler, M. Wilde, and Y. Zhao. Chimera: A Virtual Data System For Representing, Querying, and Automating Data Derivation. In *Proceedings of the 14th. Conference on Scientific and Statistical Database Management*, pages 37--46, 2002.

[50] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. GCE '08 In 2008 Grid Computing Environments Workshop, December 2008.

[51] Virtualization Technologies. Genesis Multimedia Solutions, June 2007.

[52] W. Huang, J. Liu, B. Abali, Panda, and D. K. Panda. A Case For High Performance Computing With Virtual Machines. In *ICS '06: Proceedings of the 20th annual international conference on Supercomputing, Cairns, Queensland, Australia*, pages 125--134, New York, USA, 2006. ACM.

[53] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa. Science Clouds: Early Experiences in Cloud Computing for Scientific Applications, 2008. First Workshop on Cloud Computing and its Applications, CCA'08.

[54] K. Krauter, R. Buyya, and M. Maheswaran. A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. pages 135--164, 2002.

[55] C. Mateos, A. Zunino, and M. Campo. A Survey on Approaches to Gridification. *Software Practice and Experience*, 38(5):523--556, 2008.

[56] NASA. Portable Batch System. `http://www.nas.nasa.gov/Software/PBS/pbsnashome.html`, Last visited: 16 September, 2010.

[57] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. Eucalyptus : A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Useful Systems. UCSB Computer Science Technical Report Number 2008-10, 2008.

[58] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proceedings of Cloud Computing and Its Applications*, October 2008.

[59] University of Wisconsin-Madison. *Condor Version 7.2.5 Manual*, December 2009.

[60] A. Olias. Feature - Grid in a Cloud: Processing the Astronomically Large. *iSGTW Astronomy Special Issue*, 145, October 2009.

[61] G. F. Pfister. *In Search of Clusters*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, second edition, 1998.

[62] L. Ramakrishnan, C. Koelbel, Y.-S. Kee, R. Wolski, D. Nurmi, D. Gannon, G. Obertelli, A. YarKhan, A. Mandal, T. M. Huang, K. Thyagaraja, and D. Zagorodnov. VGrADS: Enabling e-Science Workflows on Grids and Clouds with Fault Tolerance. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1--12. ACM, 2009.

[63] Cluster Resources. *Maui Scheduler Administrator's Guide*, version 3.2 edition, 2005.

[64] M. Riedel, A. Streit, Th. Lippert, and D. Kranzlmueller F. Wolf. Concepts and Design of an Interoperability Reference Model for Scientific- and Grid Computing Infrastructures. In *Proceedings of the Applied Computing Conference, in Mathematical Methods and Applied Computing*, pages 691--698. WSEAS Press, 2009.

[65] T. Rings. Testing Grid Applications Using TTCN-3. Master's thesis, Master Thesis, Institute for Informatics, ZFI-BM-2007-27, ISSN 1612-6793, Georg-August-Universität Göttingen, September 2007.

[66] R. Rivest. The MD5 Message-Digest Algorithm, 1992.

[67] D. Sommerfeld, T. Lingner, and H. Richter. Stepwise Enabling of AUGUSTUS for MediGRID. Technical Report IfI-07-12, Clausthal University of Technology, 2007.

[68] B. Sotomayor and L. Childers. *Globus Toolkit 4 Programming Java Services*. Morgan Kaufmann Publisher, 2006.

[69] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. An Open Source Solution for Virtual Infrastructure Management in Private and Hybrid Clouds. Preprint ANL/MCS-P1649-0709, July 2009.

[70] I. Sriram and A. Khajeh-Hosseini. Research Agenda in Cloud Technologies. abs/1001.3259, 2010.

[71] Nanda Susanta and Chiueh Tzi-Cker. A Survey on Virtualization Technologies. Technical report, Department of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794-4400, 2007.

[72] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A Break in the Clouds: Towards a Cloud Definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50--55, 2009.

[73] S. Venugopal, R. Buyya, and K. Ramamohanarao. A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing. *ACM Computer Survey*, 38(1):3, 2006.

[74] B. Wang, Z. Xu, C. Xu, Y. Yin, W. Ding, and H. Yu. A Study of Gridifying Scientific Computing Legacy Codes. In *Grid and Cooperative Computing - GCC 2004*, Lecture Notes in Computer Science, pages 404--412. Springer Berlin / Heidelberg, 2004.

[75] L. Wang. *Virtual Environments for Grid Computing*. University Karlsruhe press, 2008.

[76] L. Wang, G. von Laszewski, J. Tao, and M. Kunze. Grid Virtualization Engine: Design, Implementation, and Evaluation. *Systems Journal, IEEE*, 3(4):477--488, December 2009.

[77] Peter Wegner. Interoperability. *ACM Comput. Surv.*, 28(1):285--287, 1996.

[78] C. S. Yeo, R. Buyya, H. Pourreza, R. Eskicioglu, P. Graham, and F. Sommers. Cluster Computing: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers. In *Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies, Chapter 16*, pages 521--551, 2006.