First Year Cloud-like Management of Grid Sites Research Report

Henar Muñoz Frutos, Javier Martinez, Eduardo Huedo, Rubén Montero, Rafael Moreno, Ignacio Llorente, Evangelos Floros

Enhancing Grid Infrastructures with
Virtualization and Cloud Technologies

First Year Cloud-like Management of Grid Sites Research Report

Deliverable D6.3 (V1.0)
15 June 2011

Abstract

This report presents the results of the research and technological development activities undertaken during the first phase of the project by the three first tasks in which WP6 is divided.

StratusLab is co-funded by the European Community's Seventh Framework Programme (Capacities) Grant Agreement INFSO-RI-261552.

Contributors

Name	Partner	Sections
Muñoz Frutos, Henar	TID	1, 2, 4, 5, 6, 7
Martinez Elicegui, Javier	TID	1, 2, 4, 6, 7
Huedo, Eduardo	UCM	4, 5, 6
Montero, Rubén S.	UCM	4, 5, 6
Moreno, Rafael	UCM	5, 6
Llorente, Ignacio M.	UCM	5, 6
Floros, Vangelis	GRNET	3

Document History

Version	Date	Comment
0.1	09 May. 2011	Initial version for comment.
0.2	09 May. 2011	Initial version for comment.
0.3	20 Mayo. 2011	Main chapters contribution.
0.4	30 Mayo. 2011	Executive Summary, Introduction and Conclusions.
0.5	8 June. 2011	Peer review of the document.
0.6	10 June. 2011	Work on revision comments.

Contents

List of Figures

List of Tables

1 Executive Summary

Grid infrastructures are typically static, with limited flexibility for changing application parameters: OS, middleware and resources in general. By introducing cloud management capabilities, grids can become dynamic. Adding standard tools, such as virtual machines (VMs), resources can be repurposed on demand to meet the requirements of high priority applications. The cloud platform controls which application images should be running and when. This means dynamic application stacks on top of the available infrastructure, such that any physical resource can be timely repurposed on demand for additional capacity [5].

The Joint Research Activity (JRA), carried out in WP6, develops advanced technology/features for deployment on existing Cloud infrastructures through automatic deployment and dynamic provision of grid services as well as scalable cloud-like management of grid site resources. More specifically, the objectives to be accomplished can be expressed as: i) the extension of currently available open-source service-level frameworks which provide elasticity on top of cloud infrastructures, ii) the invention of new techniques for the efficient management of virtualized resources for grid services and iii) the inclusion of novel resource provisioning models based on cloud-like interfaces.

Thus, this document presents the work done in WP6 during the first year of the StratusLab project focused on the dynamic provisioning of grid services and Scalable and Elastic Management of Grid Site Infrastructure. In addition, some work has been done in the definition and implementation of Cloud-like Interfaces Specific for the Scientific Community.

Regarding the work done in the dynamic provisioning of grid service, during this year, we have deployed a grid site and provided scalability automatically for the jobs. The deployment of the grid site has involved several steps to virtualized gLite services, to configure in an automatically way each service, to specify grid site properties in a standard way, and so on. Regarding the elasticity, the number of Worker Nodes has varied according to the number of jobs. For getting this automatic mechanism, a service manager (Claudia) has been introduced in the StratusLab distribution, to manage the grid site as a whole.

For what concerns Scalable and Elastic Management of Grid Site Infrastructure, OpenNebula is extended to support the typical operations of a grid site. In particular, some virtual resource placement heuristics have been added to optimize different infrastructure metrics. Some work is being done towards cloud-aware

image management techniques by the development of a image repository in OpenNebula and support of multiple storage backends. Finally, cloud-aware network management techniques will be included.

Although according to the project's Description of Work the task of Cloud-like Interfaces Specific for the Scientific Community starts in year 2, some work has been done in order to identify and implement interfaces for Cloud resources and services. Due to it, TCloud API and OCCI are considered as the service manager and virtual machine manager API respectively and some development work is doing towards it.

Previous documents in WP6, D6.1 *Cloud-like Management of Grid Sites 1.0 Design Report* [12] and D6.2 *Cloud-like Management of Grid Sites 1.0 Software* [13], identified a set of solutions to solve the gaps identified in the deliverable D4.1 *Reference Architecture for StratusLab Toolkit 1.0* [11] and implementation work respectively. However, this document is more focused on providing information about the work done towards the dynamic deployment of grid services (sites) and mechanism for scalable and elastic management of the grid site infrastructure.

2 Introduction

The Joint Research Activity (JRA), carried out in WP6, develops advanced technology/features for deployment on existing Cloud infrastructures through automatic deployment and dynamic provision of grid services as well as scalable cloud-like management of grid site resources. More specifically, the objectives to be accomplished can be expressed as: i) the extension of currently available service-level open-source elasticity frameworks on top of cloud infrastructures, ii) the invention of new techniques for the efficient management of virtualized resources for grid services and iii) the inclusion of novel resource provisioning models based on cloud-like interfaces.

The present document comments about the work done in order towards the Cloud-like Management of Grid Sites. The developed software was already documented in the D6.2 *Cloud-like Management of Grid Sites 1.0 Software* [13], which implemented the gaps identified in *Cloud-like Management of Grid Sites 1.0 Design Report* [12].

The WP6 objectives, explained in D6.1 and developed in D6.2, include the identification, design and implementation of the technology and features for grid deployment on existing cloud infrastructures. These features move towards an automatic deployment and dynamic provision of grid services as well as scalable cloud-like management of grid site resources. To achieve these, the main work done in this work package involves:

- On top of the current virtual machine manager, the introduction of a service manager, which is able to control and configure grid services as a whole, considering the service concept as a set of virtual machines and the interconnecting network. The first example for that involves the deployment and configuration of a gLite-based grid site.
- The introduction of scalability mechanism at the service-level (as part of the service manager) to scale up and down grid services components according to some Key Performance Indicators (KPIs) and hardware usage.
- The inclusion of cloud-like interfaces based on standards to increase the interoperability in cloud service management.
- Monitoring systems at the physical, virtual hardware and grid service layers for regular administration tasks, accounting purposes and for triggering

scalability mechanisms.

- Accounting information to keep track of resource usage in order to apply site usage policies for fair share and potential resource charging.

These objectives are being implemented in the different tasks in the WP6. The task T6.1 *Dynamic Provision of Grid Services* has been working towards the provision of grid services on a cloud environment. It allows for reducing the time to execute computationally intensive services without incurring excessive costs, by the deployment of a high number of services simultaneously. This deployment requires the definition of a service hierarchy (i.e. interdependencies among service components that may need to be satisfied at deployment time only). Also, the dynamism of a grid application (*e.g.* a node fails, workload increases) can trigger service scaling, reconfiguration, or migration of that specific grid service or the whole services in the virtual cluster so that the user Quality of Experience (QoE) is not spoiled. This dynamism requires advanced monitoring techniques that deliver the required data to an intelligent element making the appropriate provisioning/reprovisioning decisions. In order to satisfy these objectives some work has been done as it is explained in Chapter 4.

- Introduction of a service manager to control the grid site as a whole
- Introduction of an elasticity framework for scaling and reconfiguration grid service components
- Grid services and grid sites considered as a service concept formalized in OVF and contextualization information
- Development of Ganglia probes for monitoring the virtual infrastructure
- Integration of Ganglia monitoring information in OpenNebula
- Collection of requirements for the integration of cloud and grid services

The T6.2 Scalable and Elastic Management of Grid Site Infrastructure will adapt an open-source Virtual Machine Manager (OpenNebula) to the typical operations of a grid site, in particular to support to define complete grid services at the infrastructure level, as a set of related virtual machines with possible deployment dependencies. In addition, virtual resource placement heuristics will be added to optimize different infrastructure metrics (*e.g.* utilization or energy consumption) and to fulfill grid service constraints (*e.g.* affinity of related virtual resources or Service Level Agreement (SLA)). The development of cloud-aware image management techniques and cloud-aware network management techniques will be included.

In order to satisfy these objectives some work has been done as it is explained in Chapter 5.

- Integration of the service manager with OpenNebula so that the VIM can support a complete grid service (set of related virtual machines with possible deployment dependencies)
- Evaluation of placement policies in OpenNebula (ongoing work).
- Development of a Image Repository in OpenNebula for image management.
- Support for multiple storage backends to access persistent images in the Image Repository (ongoing work).
- External image catalogs (ongoing work).
- VM contextualization using image information.
- Dynamic modification of "fixed" virtual networks.
- Evaluation of additional VLAN models for virtual network management (ongoing work).
- Automatic setup of simple TCP/UDP firewall rules for VMs (ongoing work).
- VM contextualization using virtual network information.
- Improved fault tolerance in OpenNebula.
- Grouping of physical hosts in clusters.

The Task 6.3 Cloud-like Interfaces Specific for the Scientific Community will define the cloud interfaces for the system to provide a Grid as a Service. To provide such a grid or Cluster as a Service interface the following extensions to current cloud approaches has to be considered: i) the ability to specify a grid service as a whole, ii) the ability to specify the characteristics of each component of a grid service including virtual networks and storage, iii) the ability to specify context information (*e.g.* CA certificates or component role) to each service component so it can be integrated within other grids and the integration of Grid services within the IaaS interface.

In order to satisfy these objectives some work has been done as it is explained in Chapter 6.

- Grid service as a whole specification by OVF
- Specification of characteristics of each component of a grid service including virtual networks and storage by OVF
- TCloud as the Claudia API
- Enhancements in OGF OCCI implementation in OpenNebula.
- Authentication in OpenNebula based on LDAP and grid/VOMS certificates.
- Authorization in OpenNebula based on groups and roles (ongoing work).

2.1 Organization of Following Chapters

The document is organized as follows: Chapter 3 exposes some requirement from the experimentation of the usage of grid services. Chapter 4 explains the work done in the task Dynamic Provision of Grid Services, the Chapter 5 the summary of the work in the Scalable and Elastic Management of Grid Site Infrastructure task and the Chapter 6 works on Cloud like-Interfaces Specific for the Scientific Community. Finally, the Chapter 7 provides some conclusions.

3 Grid services requirements

In this section we briefly summarize on the findings from our activities in the context of WP5 concerning the deployment and operation of a production grid site on the StratusLab reference cloud service. This activity has provided a valuable input for WP6 regarding important extensions and interface integration that can facilitate the optimal exploitation of cloud computing capabilities from grid services.

Grid site administrators are considered as a regular IaaS clients from the cloud layer point of view. As such, they are fully responsible for the proper deployment of a grid site, for the configuration of the various the services, the allocation CPU and storage capacity for the VOs that the site supports, to monitor the operation of the site and keep accounting information for its usage. In the typical case that the site is operated as a part of a distributed grid infrastructure, as it is the case for instance for the EGI sites, the grid administrator is responsible to keep the site up to date with the evolution of the operational policies defined by the infrastructure, and also ensure the proper integration of the local site with the central services that for example aggregate accounting and monitoring information for all the sites.

On first approach, the provision of grid services on top of cloud technologies could be considered just another application for cloud services. Indeed, one can operate a grid site using virtualization technologies without any particular integration between the two layers. Nevertheless, we believe that in order for grid services to fully exploit the potential of cloud computing there should be a bridging of these two worlds on a technical and operational level. In the following paragraphs we take a closer look on the requirements and integration potentials that apply both the cloud and grid services.

3.1 Grid Site Deployment

The installation and configuration of grid site still remains a nontrivial and time consuming activity. IaaS clouds can alleviate this by providing pre-configured VMs with the grid software already pre-installed. The requirements here mostly target the Grid computing community (EGI and EMI) that should be responsible for the proper creation of grid node VM appliances, their provision from open distribution channels (like the StratusLab Marketplace) and their maintenance by releasing for example regular updates that incorporate new grid software releases, appliance bug fixes and security improvements.

The provision of grid services on top of cloud computing environments impacts many of the configuration parameters typically declared during the phase of site deployment. Many of these parameters become irrelevant or cannot expose proper information about the site structure and capabilities. These parameters are for example the hardware profile of the site, which typically focuses on the low level cpu and memory details of the Worker Nodes and the information regarding site locality (e.g. Latitude & Longitude of the site installation). Overall, the grid middleware today follows a static approach for defining grid site attributes. Optimally the configuration tools from the side of the grid will evolve and adapt to the dynamic and virtualized nature of cloud infrastructures.

The cloud service can contribute significantly in the deployment process by providing customized tools that automate the process of grid-site definition and instantiation. In the context of StratusLab the latter is achieved through the usage of Service Layer and the employment of OVF files for the definition of grid-site structure, initialization parameters and expected elasticity behavior.

Part of the deployment process is the issuing of grid certificates for those services that require SSL-based authentication and encryption. Such services are running for instance in the CE, SE and APEL nodes. These certificates have to be bound to specific respective DNS names included in the certificate CN (Common Name) field. In a dynamic cloud environment typically the DNS of a VM is not known until it is instantiated by the cloud service. Since re-generation of grid certificates to bind to a new DNS name is a time consuming process, the cloud service should provide an advanced IP reservation facility that will allow grid administrators to get hold of a specific IP address that no-one else can use in the cloud. This IP has to be passed as a parameter upon VM instantiation instructing the cloud service to assign this specific IP to the new VM instance. This requirement has already been identified in the context of StratusLab and has been enabled in the VIM (i.e. OpenNebula) and Service Management layers.

3.2 Grid Site Operation

We consider three aspects of a grid site's operation that can be influences by cloud services: monitoring, accounting and site elasticity.

3.2.1 Monitoring

Monitoring has been identified as one of the subjects that are difficult to concretely define. This is because the process of monitoring can cover multiple requirements, interact with a different number of sub-systems in a given architecture and target various actors. Thus the notion of completeness of a monitoring component for a specific architecture is subjective and depends on the scope of this architecture. In general a monitoring system is used in order:

- To ensure the system's health (i.e. all sub-systems are functioning properly)
- To facilitate the tracking of security problems and system breaches

Within StratusLab we have already identified three layers where monitoring could be applied:

- The physical layer. For monitoring of the physical nodes hosting the cloud services including storage and networking.
- The cloud layer. For monitoring the status and health of the VIM and of the VM instances
- The application layer. Which provides application specific monitoring capabilities. For example here would reside the monitoring functionality installed by a grid site.

As a concrete example, in the context of StratusLab we have used and extended the Ganglia monitoring system in order to monitor the physical layer and part of the cloud layer (i.e. the probes developed that communicate with libvirt in the hosting nodes). WP4 has developed a simple web monitoring tool that covers the monitoring requirements for the cloud layer. This monitoring tools have been deployed and extensively used by WP5 for the provision of the reference cloud service. Finally in the grid layer, our production grid site is connected with the centralized monitoring service of EGI called gstat [1] and the Greek-NGI one [2] which is based on Nagios and provides a generic overview of the site's status. Our experience and the interaction we had with other projects, indicates that the existing tools and approaches for monitoring are sufficient and viable for future demands.

The two first layers could be further integrated giving a unified on-stop-shop monitoring facility for the infrastructure covering both the physical and cloud layer.

The application/grid layer on the other hand should remain independent for at least two reasons:

- Cloud service providers shouldn't be responsible for monitoring individual applications
- Cloud service providers could be potentially restricted by law to monitor and tamper applications, data and network traffic.

3.2.2 Accounting

Like monitoring, accounting is also a loosely defined service that may be used to cover different requirements from the infrastructure point of view:

- Collect information that will drive the billing policy of the service
- Collect usage information in order to ensure fair share of the resources

[1] http://gstat-prod.cern.ch
[2] http://nagios.hellasgrid.gr

- Collect information that will direct scheduling and placement policies for workload management and VM instantiation

Once again the two worlds of cloud and grid services utilize separate accounting components that collect overlapping information.

So far the grid accounting system has been mainly designed to support the collection of basic statistics. If we consider though a typical scenario where commercial cloud providers will offer resources to scientists or hybrid clouds where government funded clouds will burst to commercial clouds in order to handle peek workloads, it is crucial that the accounting system collects detailed usage information on the VO and individual user levels.

In our production site we have used glite-APEL for site accounting. APEL collects only a limited amount of information such us the number of jobs submitted or total CPU time consumed per site/user/VO. Integration with cloud services will require a much more detailed report, including network bandwidth, storage space, IP addresses, and potentially use of software licenses (wherever applicable).

On the other hand the cloud layer should be able to re-use this information in order to charge costs or/and to enforce quotas. Of course this requires that the cloud accounting sub-system is keeping a complete set of information regarding the above metrics. Also the data provided from the grid layer should be matched with those on the cloud layer in order to produce a unified accounting record either per grid VO, grid user or both.

3.2.3 Grid Site Elasticity

Resource elasticity is one of the most well known advantages introduced by cloud computing. Grid sites should be able to capitalize on this by being able to dynamically adjust their dimensions based on fluctuating demands. Typical dimensions of a grid site are:

- Processing capacity: being able to modify the size of the cluster by adding or removing WNs on demand.
- Processing capability: being able to modify the profile of the WNs by adding new ones with more CPU cores, local storage and memory.
- Storage capacity: being able to modify the available storage provided by the SE node.

In order to implement such elasticity functionality the grid and cloud layer should work closely together. The control and definition of elasticity rules and triggers have to be defined in the cloud layer, since this is the place where virtual resources are controlled. The grid layer should provide all the necessary information about the current state of grid resource utilization (e.g. number of jobs in the CE) since this is the place where all information about grid resources are kept.

This dynamic behavior of grid sites on the other hand may cause inconsistency on the global level if the information about the sites new capabilities are not

announced promptly to the top level information systems (e.g. top-level BDII), causing job management services like the WMS (Workload Management System) to make inappropriate job scheduling decisions. Thus also the grid services should be able to cope with dynamic grid sites whose dimensions and capabilities are continuously changing.

4 Dynamic Provision of Grid Services

The task 6.1 *Dynamic Provision of Grid Services* has been working towards the provision of grid services on a cloud environment. It allows for reducing the time to execute computationally intensive services without incurring excessive costs, by the deployment of a high number of services simultaneously. This deployment, commented in section 4.2, requires the definition of a service hierarchy (i.e. interdependencies among service components that may need to be satisfied at deployment time only) which is specified in the grid site specification (see section 4.1). Also, the dynamism of a grid application (e.g. a node fails, load grows) can trigger service scaling, commented in 4.3, reconfiguration, or migration of that specific grid service or the whole services in the virtual cluster so that the user Quality of Experience (QoE) is not spoiled. This dynamism requires advanced monitoring techniques that deliver the required data to an intelligent element making the appropriate provisioning/reprovisioning decisions.

4.1 Grid Site Specification

As commented in D6.1 [12], there is a need for a standard language to define the services to be deployed in the Cloud. In that document the Open Virtualization Format (OVF) was selected for that aim. OVF's objective [4] is to specify a portable packaging mechanism to foster the adoption of Virtual Appliances (VApp) (i.e. pre-configured software stacks comprising one or more virtual machines to provide self-contained services) as a new software release and management model (e.g. through the development of virtual appliance lifecycle management tools) in a vendor and platform neutral way (i.e., not oriented to any virtual machine technology in particular). OVF is optimized for distribution and automation, enabling streamlined installations of VApps.

Figure 4.1 shows the structure of an OVF package. This file includes an OVF descriptor (an XML file describing the VApp), resources used by the VApp (virtual disk, ISO images, internationalization resources, etc.). The OVF descriptor is composed by a set of sections:

- Virtual System Section: It involves information about each virtual machine with virtual hardware information, the software installed and so on.
  `<VirtualSystem ovf:id="workernode">`
- Virtual Hardware Section: It contains virtual hardware resources required in

each virtual machine. This hardware information is specified by using the Item element. The ResourceType specifies the type of resource, 4 for RAM, 3 for CPU and 10 for disk

```
<VirtualHardwareSection>
<Info>Virtual Hardware Requirements:  512Mb, 2 CPU </Info>
<Item>
<rasd:Description>Number of virtual CPUs</rasd:Description>
<rasd:ResourceType>3</rasd:ResourceType>
<rasd:VirtualQuantity>2</rasd:VirtualQuantity>
</Item>
<Item>
<rasd:AllocationUnits>MegaBytes</rasd:AllocationUnits>
<rasd:Description>Memory Size</rasd:Description>
<rasd:ResourceType>4</rasd:ResourceType>
<rasd:VirtualQuantity>512</rasd:VirtualQuantity>
</Item>
```

- Product Section: This section involves information about the software installed in the VM. It can include configuration parameters needed by the software.

```
<ProductSection ovf:class=s̈tratus.glite.grid.wns̈
<Info>The gLite 3.2 packages for the Worker Node have
been installed in this image along with
the glite MPI_utils metapackage providing support for
OPENMPI and MPICH2</Info>
<Product>gLite 3.2</Product>
<Version>1.0</Version>
<Property ovf:key="VO" ovf:value="vo.stratuslab.eu">
<Property ovf:value="vm047.ypepth.grnet.gr" ovf:key="HOSTNAME"/>
```

- Network Section: Information about the network (public and private) in the service.

```
<NetworkSection>
<Info>Network information</Info>
<Network ovf:name="glite_public" rsrvr:public="true">
<Description>Public Network</Description>
</Network>
</NetworkSection>
```

- Disk Section: Information about the disk which are incorporated in the different VMs.

```
<References>
<File ovf:id="computingelement" ovf:href="
```
http://appliances.stratuslab.eu/images/grid/ce/sl-5.5-x86_64-grid.ce/1.0/sl-5.5-x86_64-grid.

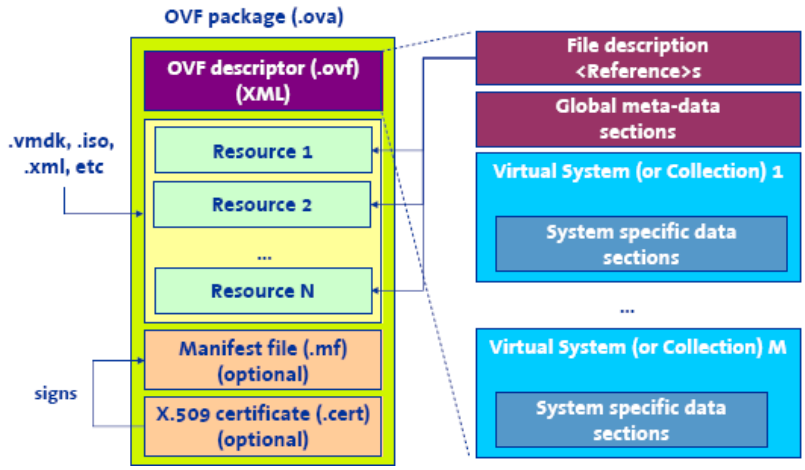

Figure 4.1: *OVF package and descriptor structure*

```
ce-1.0.img.gz"/>
</References>
<DiskSection>
<Disk ovf:diskId="storageelement" ovf:capacity="1536MB"
ovf:fileRef="storageelement"/>
</DiskSection>
```

4.1.1 OVF for specifying Grid sites

Thus, OVF specifies how to package and distribute software to be run in virtual machines. Our contribution is focused on utilizing OVF as basis for a service definition language for deploying complex Internet applications in a StratusLab grid-cloud infrastructure. These applications consist of a collection of virtual machines (VM) with several configuration parameters (e.g., hostnames, IP addresses and other application specific parameters) for software components (e.g., web server, application server, database, operating system) included in the VMs. Most of these parameters are unknown before the deployment time because the service provider does not know particularities of the IaaS cloud or datacenter in which the deployment takes place.

Thus, taking into account the VM generated in the grid site, we have three types:

- Compute Element (CE) [8]: The gLite 3.2 packages for CREAM Computing Element (CREAM CE) have been installed in this image along with the glite_MPI_utils metapackage providing support for OPENMPI and MPICH2.
- Storage Element (SE) [9]: The gLite 3.2 packages for Storage Element are installed in this image.

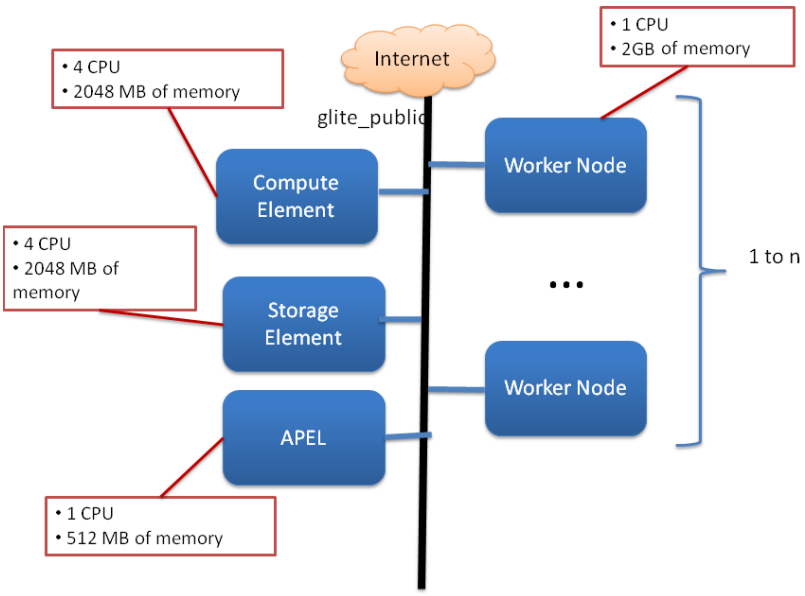

Figure 4.2: *Grid site Structure*

- Worked Node (WN) [10]: The gLite 3.2 packages for the Worker Node have been installed in this image along with the glite_MPI_utils metapackage providing support for OPENMPI and MPICH2
- Accounting Processor for Event Logs (APEL) [7]: This image contains the gLite 3.2 packages for instantiating an APEL node, which is used to collect and broadcast accounting information for a given grid site.

It is possible to see in Figure 4.2, that the VM instances are connected to a public network. This means that:

- The grid site OVF is composed by 4 VirtualSystem, one each VM type we have (CE, SE, WN, APEL)
- The grid site OVF has a public network where all the VM are connected to
- Each VM has a disk to mount the image to be deployed
- The hardware requirements are specified in the figure.
- The configuration parameters are specified in the Product Section of each VM.
- The Startup section specifies the order to boot the VMs (firstly the WN, then SE and finally CE).

4.1.2 OVF extensions required for the grid site specification

The OVF provides more of the requirements we need by still there are some pending issues which are not covered. OVF can be extended easily when new extensions are required. This extension can be provided by:

Scalability Scalability rules to be part of the OVF, to specify when the service can scale up and down and under which conditions

KPI Information about the KPI which drives the scalability (it can be a service KPI or a hardware KPI (e.g. CPU).

Number of replicas to be deployed The number maximum and minimum of the replicas to be scale up and down.

4.2 Grid Site Deployment

StratusLab provides mechanisms for the automatic deployment of grid sites by the introduction of the service manager (Claudia) in the StratusLab distribution.

The Claudia platform [15] is an advanced service management toolkit that allows service providers to dynamically control the service provisioning and scalability in an IaaS Cloud. Claudia manages services as a whole, controlling the configuration of multiple VM components, virtual networks and storage support by optimizing the use of them.

The grid site, specified in the format commented in Section 4.1, can be deployed in the Cloud with the help of the service manager. By using the deployment functionality, which Claudia provides (following the TCloud specification) [14], the grid site can be deployed with the images specified in the OVF. Claudia processes the OVF and does the needed requests to deploy networks and virtual machines to the Virtual Machine Manager (OpenNebula). The configuration information arrives at the VM by using the contextualization. By using scripts and contextualization information, all VMs execute the YAIM tool [16] against a set of inferred configuration files. As a result, all the VMs from the grid site have been deployed in the Cloud are interconnected by a public network, configured correctly and the grid site is up and running.

However, some work has been done before the deployment. It has involved:

- Generation of the Grid site appliances VMs (CE, SE, WN, APEL) located in the StratusLab appliance repository [8], [9], [10] and [7].
- Definition of the OVF, as explained in Section 4.1, with all the virtual hardware requirements and static configuration information
- Development of scripts to configure the grid site from the information offered in contextualization
- Some changes in the Claudia code to satisfy grid site requirements (static IP support, definition of contextualization files in the OVF, and so on).

4.2.1 Grid site Contextualization

Contextualization is defined as the process by which a virtual machine instance is configured [12]. In general, contextualization consists of passing arbitrary data to the virtual machine at boot time. The information can involve network contextualization and software contextualization.

4.2.1.1 OpenNebula Contextualization

The method we provide to give configuration parameters to a newly started virtual machine is using an ISO image (OVF recommendation). This method is network agnostic so it can be used also to configure network interfaces. In the VM description file you can specify the contents of the ISO file (files and directories), tell the device the ISO image will be accessible and specify the configuration parameters that will be written to a file for later use inside the virtual machine [**?**].

In VM description file you can tell OpenNebula to create a contextualization image and to fill it with values using CONTEXT parameter. For example:

```
CONTEXT = [
hostname = "vm037.one.ypepth.grnet.gr",
ip_private = "10.95.34.78",
files = "/service/init.sh /service/certificates
/service/service.conf",
target = "sdc" ]
```

4.2.1.2 Software Contextualization

The OVF language can also provide a mechanism to provide the contextualization information for the software installed in the VMs. The software configuration information can be provided as a `Property` in the `ProductSection` in the OVF file.

This information is processed by Claudia, which in deployment time generate the OVF environment file per virtual machine. The OVF environment [4] is a format which is used by the deployment platform to provide configuration parameters to guests at deployment time. Claudia is able to fill automatically these parameters during the boot process. The URL where the OVF environment file (ovf-env.xml) is located is provided to the ONE template by a URL.

```
<ns1:Environment ns1:id="workernode">
<ns1:Property ns1:key="VO" ns1:value="vo.stratuslab.eu"/>
<ns1:Property ns1:key="HOSTNAME" ns1:value="vm037.one.ypepth.grnet.gr"/>
```

Besides the ovf-env.xml file with all the configuration information, it is required a script which is able to parse the file and configure the different glite VMs in the right way. It can be passed also as a contextualization information.

```
CONTEXT = [
CustomizationUrl="http://84.21.173.28:18888/grnet.
   customers.glite.services.gridsite.vees.CE.replicas
   .1",
```

```
files="/home/tcloud/tcloud-server/repository/grnet.
   customers.glite.services.gridsite.vees.CE.replicas
   .1/ovf-env.xml",
target="hdc" ]
```

In addition, Claudia should pass to OpenNebula all the scripts required to configure the grid site by the information in the ovf-env.xml. This is the ovf-env parser (see Section 4.2.1.3).

```
CONTEXT=[files="/home/claudia/extraFiles/OVFParser.py"..]
```

Finally, all the certificates needed for configuring the grid site should be transferred also from Claudia to the VM in contextualization.

```
CONTEXT=[files="path/certificate1"..]
```

4.2.1.3 ovf-env parser

The grid site configuration information arrives at the VM as part of the ovf-env.xml file which is included in the mounted disk by contextualization. The ovf-env.xml file should be parsed to generate the required yaim configuration files during the VM contextualization. The ovf-env parser code is in charge of it.

4.2.2 Claudia implementation changes

A requirement taken from the grid site is the needs for static IPs. Some VMs in the grid site require a digital certificate which corresponds to a concrete IP. This certificate has a duration of a year for the duration of which we have to keep the associated IPs. Due to this, Claudia should manage the static IP for customers and specify a way so that the user can utilize to configure the grid site in the OVF.

As said before some scripts and certificates should be passed to the VM during contextualization. Thus, there is a need for specifying all the files to be passed in contextualization in the OVF and use the OpenNebula contextualization mechanism to provide it. The solution is to provide these scripts, certificates, and so on, as a property in the OVF. Claudia processes this information and includes it in the template ONE.

```
<Property ovf:key="SCRIPT_LIST" ovf:type="string"
ovf:value="script1.shscript2.sh">
```

4.2.3 Grid site deployment introducing Claudia

Figure 4.3 shows a high level flow describing the deployment of the grid site into the StratusLab platform. The main steps are:

1. As a previous step before the beginning of the deployment process, the Service Provider must define the grid site structure, conditions and characteristics in the OVF file. It includes the description of all the grid site's VMs (OS, hardware characteristics, networks, etc.). In this case, the service comprises three VMs (see Figure 4.2): the CE implements the Compute Element, the SE the Storage Element, the APEL the Accounting Processor for Event Logs

and the WN the Worker node. All components are interconnected through a public network called *glite_public*.

2. In addition, the grid site user has to generate all the VMs involved in the services and store them into the Appliance Repository.

3. At this point, the grid site user is ready to request the service deployment. The interface through the Service Manager (SM) implements the TCloud API for supporting the needed operations at this level. The deployment request will include the OVF descriptor.

4. The SM revises the OVF file searching useful information for the service lifecycle management. like the startup order of all the VMs of the Service, scalability rules, resource requirements... The SM also translates the OVF file into the needed OpenNebula templates (for both networks and VMs).

5. The next step is to provide the needed networks and to start the deployment of VMs in the right order, by interacting with OpenNebula.

6. At this point, the OpenNebula engine has all the information to deploy the service into the StratusLab Platform. It localizes the images in the Appliance Repository and installs them in the corresponding resources.

4.3 Grid Site Scalability

Grid applications deployed over cloud technologies should benefit from scalability at service level, which conceals low level details from the user. In the service multilayer, as we have, where the service is composed by a set of VMs (CE, SE, APEL and WN), StratusLab can provide scalability in each layer in the service, in case it is required.

Claudia, the service manager, manages the service monitoring events and scalability rules. In addition, it is responsible for dynamically asking for virtualized resources to a virtual machine manager like OpenNebula, trying to avoid over/under provisioning and over-costs based on SLAs and business rules protection techniques.

Thus, Claudia provides a means for users to specify their application behavior in terms of adding or removing more of the same software or hardware resources [1] by means of *elasticity rules* [2]. The elasticity rules follow the Event-Condition-Action approach, where automated actions to resize a specific service component (e.g. increase/decrease allocated memory) or the deployment/undeployment of specific service instances are triggered when certain conditions relating to these monitoring events (KPIs) hold.

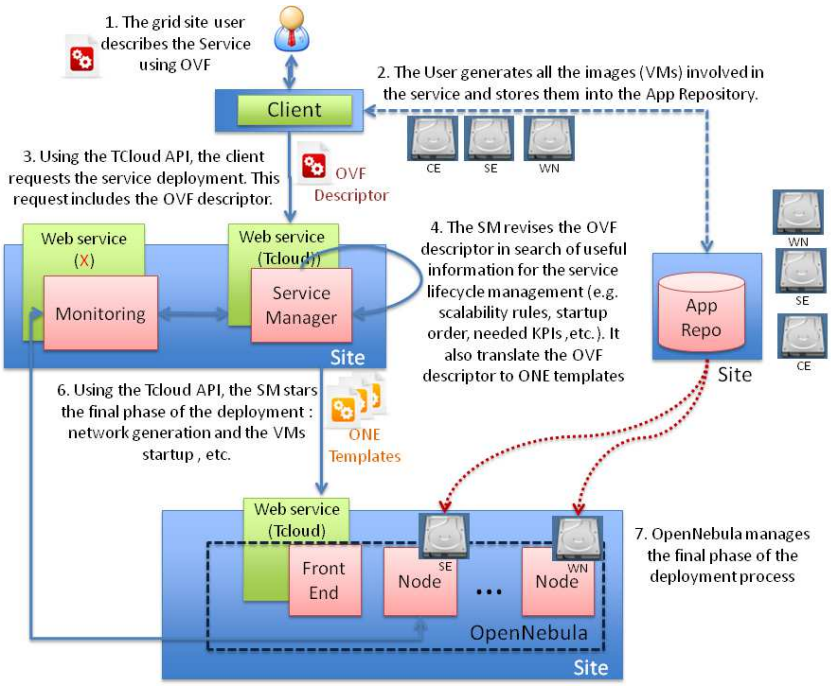

Figure 4.3: *Deployment Scenario*

4.3.1 Scalability Information in the OVF

4.3.1.1 KPI

To enable the definition of custom Key Performance Indicators (KPIs) we added a section `KPIsSection`. The section enables the Customer to define its own set of KPIs. The KPIs will be supplied by the grid user and the values specified in the KPIsSection will be passed to Claudia for the monitoring probe.

The KPIsSection contains a list of `KPI` tags, each with a KPIname attribute (describing the name of the KPI). The KPIname has to be unique, in order to identify the service and avoid conflict with other KPIs. Also it contains a KPItype when the scalability is due to virtual hardware information. An example of the KPIsSection is shown below.

```
<rsrvr:KPISection>
<ovf:Info>KPIs Section</ovf:Info>
<rsrvr:KPI KPIname="jobqueueutilization"/>
</rsrvr:KPISection>
```

For the grid site the KPI used is the jobqueueutilization which means the utilization percentage of the job queue. This is defined by the following equation:

$$jobqueueutilization = \frac{number\_of\_jobs\_running}{total\_cpu\_slots}$$

As it is a percentage the KPI value should be:

$$0 \leq KPI \leq 1$$

4.3.1.2 Elasticity rules

Standard OVF is limited to the fixed-size deployments and does not take dynamic scaling into account. This is a consequence of the `VirtualSystem` tag semantics in standard OVF, where each one of those tags represent exactly one virtual machine.

In order to govern the elasticity, a simple section has been defined, `ElasticityArraySection`, consisting of a sequence of `Rule` tags. Within each `Rule` the following information is specified:

- KPI name. The KPI identifier whose value governs the scalability of the array. This KPI has to be defined within the `KPIsSection` described in Section 4.3.1.1.
- Windows. The time interval for sampling the KPI. The semantics of this window are that of a sliding window, e.g. 5 minutes.
- Frequency. The amount of samples to take in the window. For example, 60 samples per window, that is 6 samples per minute (1 sample each 10 seconds) for a 10 minutes window.
- Quota. The normalized "quantity of KPI" that a single VM instance can sustain in a steady state. The average KPI in the window will be compared with the defined quota multiplied by the number of currently active VMs in order to decide if the array has to be expanded or shrunk.

```
<rsrvr:ElasticArraySection>
<Info>
There are two elasticity rules, to scale up and down the
WN components
</Info>
<rsrvr:Rule>
<rsrvr:KPIName>jobqueue</rsrvr:KPIName>
<rsrvr:Window unit="minute">5</rsrvr:Window>
<rsrvr:Frequency>20</rsrvr:Frequency>
<rsrvr:Quota>1</rsrvr:Quota>
<rsrvr:KPIType>agent</rsrvr:KPIType>
</rsrvr:Rule>
</rsrvr:ElasticArraySection>
```

This means that the threshold to scale up T_u and to scale down T_d can be obtained as:

$$T_u = quota * \frac{1 + tolerance}{100}$$

and

$$T_d = quota * \frac{1 - tolerance}{100}$$

4.3.2 Probe development

As said before, the grid site scalability is driven by the number of jobs which the CE is processing at a given moment. This requires, the continuous monitoring of the queues in the CE. For this purpose a python script is being developed that will be used for monitoring the number of available and total slots in a grid site. The script polls this information from qstat and pbsnodes command line tools of the TORQUE resource manager [3] and communicates them to the service manager using the latter's RESTful API.

4.3.3 Load Balancer Support

Essentially the probe described in the previous paragraph is responsible for the calculation of the scalability KPI and for passing it to the service manager. Based on this KPI the service manager makes the decision of scaling the site up or down depending on the grid administrator defined rules and thresholds. This is achieved by the close cooperation between the service manager and the load balancer (LB) component that is running as a service on the CE. LB is running constantly waiting for notifications from the Service Manager. As a service it provides two functions:

addWorkerNodes This function is invoked by the Service Manager in order to notify the LB that a new set of WNs have been instantiated. The LB takes care to update the list of target nodes for LRMS in order new jobs to be scheduled there.

removeWorkerNodes During the scale down phase, unused WNs should be removed from the queuing system. This entails shutting down the WN VMs from the Service Manager and removing the nodes from LRMS. The number of decommissioned WNs is defined by the Service Manager but the decision regarding the actual nodes to be removed is taken by the LB itself. This is because the WNs to be removed shouldn't have any jobs running on them and this is an information one can get from within the CE by appropriately probing the queues using qstat and pbsnodes commands. Thus the Service Manager only needs to pass the number of WNs for removal and the LB services returns the hostnames of the nodes removing them in parallel from the LRMS.

4.3.4 Grid site scalability introducing Claudia

As commented previously, the grid site needs to scale the worker nodes according to the number of jobs. Thus, Service Providers define elasticity rules in the service manifest to assure the right operation of their services and/or the final user experience. In this case, the virtual machine to be scaled is the WN, where new instances are deployed to avoid system overload:

```
<VirtualSystem ovf:id="WN" rsrvr:min="1" rsrvr:max="8"
rsrvr:initial="1"> <VirtualSystem ovf:id="CE" rsrvr:min="1"
rsrvr:max="1" rsrvr:initial="1">
```

Moreover, the Service Provider has defined an elasticity rule which adds a new instance of the WN to avoid the system overload. The KPI used to evaluate this rule is the jobqueue utilization, i.e., the number of running jobs. Thus, if the KPI exceeds a given threshold, a new WN is deployed.

Figure 4.4 shows how the StratusLab platform manages the scalability of the grid site.

The main steps are:

1. The probe in the CE provides KPIs values for the service to the SM.
2. The Service Manager evaluates, in real time, the elasticity rules of the grid site. When the corresponding KPI exceeds the given threshold, the SM starts the deployment process of a new WN instance.
3. Using the TCloud API, the SM requests for a new replica to be deployed. The steps here are the same as in the Deployment Section 4.2.3.

4.4 Advanced Monitoring Techniques

Ganglia is a scalable distributed system monitor tool for high-performance computing systems. It allows the user to remotely view live or historical statistics for all machines that are being monitored. The presentation is done through a web front-end that provides a view of the gathered information via real-time dynamic web pages. Ganglia makes the monitoring more efficient and scalable in big installations. Also, this monitoring system can be easily extended to monitor other aspects of the host fabric such as the number of VMs running in a specific host or information about them. Therefore, we decided to use Ganglia as the monitoring tool both for the physical and virtual infrastructure, as well as to feed OpenNebula with monitoring information.

4.4.1 Development of Ganglia Probes for Monitoring the Virtual Infrastructure

A couple of Ganglia probes have been developed to obtain metrics directly from the hypervisors in order to monitor the deployed virtual infrastructure. Since

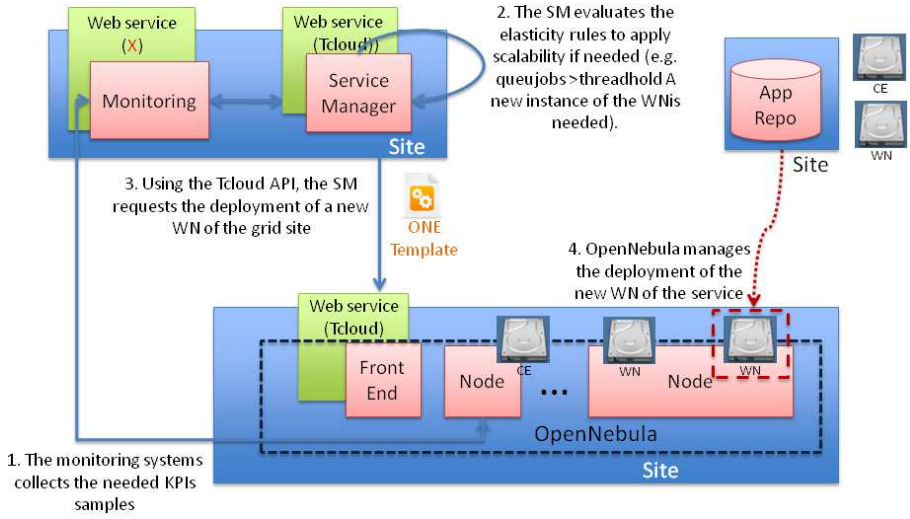

Figure 4.4: *Scalability Scenario*

VM information is not gathered by Ganglia, a script is provided to get that information. The same probe that gets information for Xen and KVM is used to get this data and push it to Ganglia. This probe has to be copied to each of the nodes, or put in a path visible by all the nodes. The information obtained by these probes needs to be periodically pushed to Ganglia in a metric called `OPENNEBULA_VMS_INFORMATION`. The `gmetric` command and the cron subsystem are used for this.

To make it refresh automatically, this command should be added to the cron subsystem. Executing it every minute is usually fine. Any other information can be pushed using any metric starting with `OPENNEBULA_`. New metrics defined this way would allow the definition of new virtual resource allocation heuristics, as described in section 5.1.

4.4.2 Integration of Ganglia Monitoring Information in OpenNebula

A new information driver has been developed to obtain monitoring information from Ganglia, instead of executing probes in the hosts using ssh. This information manager connects to Ganglia and gets the monitoring values listed in Table 4.1.

Moreover, it will also get any metric from Ganglia that starts with `OPENNEBULA_`, and will add it to the monitoring data pushed to OpenNebula as the name of the metric without that prefix. For example, if we add to Ganglia the metric `OPENNEBULA_DISK_FREE`, that data will be pushed to OpenNebula as `DISK_FREE`. It is also possible to push metrics with the same name as the standard metrics in the previous table, that way the user can change the way those metrics are computed. For example, `OPENNEBULA_TOTALCPU` will be used instead of the inter-

Table 4.1: *Ganglia and OpenNebula metrics*

Ganglia metric	OpenNebula metric
cpu_num	TOTALCPU
cpu_speed	CPUSPEED
mem_total	TOTALMEMORY
mem_free	FREEMEMORY
cpu_idle	FREECPU
bytes_out	NETTX
bytes_in	NETRX

nal TOTALCPU computation it is defined.

5 Scalable and Elastic Management of Grid Site Infrastructure

The T6.2 Scalable and Elastic Management of Grid Site Infrastructure will adapt an open-source VIM (Virtual Infrastructure Manager), OpenNebula [6], to the typical operations of a grid site, in particular to support to define complete grid services at the infrastructure level, as a set of related virtual machines with possible deployment dependencies. In addition, virtual resource placement heuristics will be added to optimize different infrastructure metrics (e.g. utilization or energy consumption) and to fulfill grid service constraints (e.g. affinity of related virtual resources or SLA). The development of cloud-aware image management techniques and cloud-aware network management techniques will be included.

5.1 Virtual Resource Placement Heuristics

5.1.1 Evaluation of Placement Policies in OpenNebula

In OpenNebula, it is possible to implement several placement policies by carefully choosing the `RANK` expression. Each VM has its own rank and so its own policy, therefore different policies can be applied to different instance types. Possible policies are:

- Packing Policy: It is aimed at minimizing the number of cluster nodes in use. The heuristic used is to pack the VMs in the cluster nodes to reduce VM fragmentation, therefore those nodes with more VMs running are used first. For example:

  ```
  RANK = RUNNING_VMS
  ```

- Striping Policy: The objective is to maximize the resources available to VMs in a node. The heuristic used is to spread the VMs in the cluster nodes, therefore those nodes with less VMs running are used first. For example:

  ```
  RANK = "- RUNNING_VMS"
  ```

- Load-aware Policy: The objective is to maximize the CPU available to VMs in a node. The heuristic is to use those nodes with less CPU load, therefore those nodes with more free CPU are used first. For example:

  ```
  RANK = FREECPU
  ```

More policies will be developed and evaluated using heuristics based on the information gathered from new monitoring metrics gathered as described in section 4.4. For example, the packing policy could be combined with a technique to selectively power on and shut down machines to achieve energy savings.

5.2 Cloud-Aware Image Management Techniques

5.2.1 Development of a Image Repository in OpenNebula for Image Management

The Image Repository system allows OpenNebula administrators and users to set up images, which can be operative systems or data, to be used in VMs easily. These images can be used by several VMs simultaneously, and also shared with other users. There are three different types of images in OpenNebula:

- OS: A working operating system. Every VM template must define one DISK referring to an image of this type.
- CDROM: Readonly data. Only one image of this type can be used in each VM template.
- DATABLOCK: A storage for data, which can be accessed and modified from different VMs. These images can be created from previous existing data, or as an empty drive. VM templates can use as many datablocks as needed.

Users can manage the image repository from the OpenNebula command line interface with the `oneimage` command. Users can create new images specifying an image template. Another way to create a new image is copying it from an existing one. To do this, the user has to mount the original image in a VM template, and tell OpenNebula to save the image in a different onei (using the `save_as` operation). OpenNebula will save the changes made to the source image as a new one when the machine is shut down. The image owner –or the OpenNebula administrator– can delete an image from the repository.

An image can be *public* for every user to use in their VMs, or *private*. Public images are always cloned before being used. An image can be also *persistent* or not. Persistent images are never cloned but rather used from the original image. Therefore, an image cannot be public and persistent at the same time.

5.2.2 Support for Multiple Storage Backends to Access Persistent Images in the Image Repository

VM disk images can be provisioned using two approaches: block devices and files. The image repository has been architected to support these two approaches and to easily incorporate different technologies in each area. The first drivers support file-based VM disks of any type (persistent and volatile, OS and data disks, and public and private images). These set of drivers are targeted at distributed file-systems that implements POSIX interfaces like GlusterFS or Lustre among others.

5.2.3 External Image Catalogs

The Image Repository serves as a natural place to store the images within a site, usually managed as any other data repository (e.g. backups). However in a federated environment, and specially in Grids, there is the need of sharing VM images across sites. An image catalog is a collection of VM images exposed remotely that are downloaded, adapted and registered to run in a given site. In this case the image repository would act as a cache of the remote catalogs. We are exploring this hybrid storage model to integrate VMs from StratusLab services (Marketplace) as well as third-party providers (Amazon S3).

5.2.4 VM Contextualization Using Image Information

Using `CONTEXT` parameter in the VM description file, it is possible to tell OpenNebula to create a contextualization image and to fill it with values. Variables inside `CONTEXT` section will be added to `context.sh` file inside the contextualization image. A new method has been added to specify these variables using variables from the image template:

- `$IMAGE[<image_attribute>, IMAGE_ID=<img_id>]`: Any single value variable in the image template, like for example:

  ```
  root = "$IMAGE[ROOT_PASS, IMAGE_ID=0]"
  ```

5.3 Cloud-Aware Network Management Techniques

5.3.1 Dynamic Modification of "Fixed" Virtual Networks

Virtual networks of type "fixed" have been improved to allow its dynamic modification without the need of re-creating them. This will potentially support an elastic management of public IPs as follows: each user is assigned with a private set of IPs obtained from the site's public IP pools. Then, VMs can request specific IPs from the elastic set of the user. The sysadmins of a site can add or remove IPs from the user's network using this new functionality, so implementing elasticity for the public addresses.

5.3.2 Evaluation of Additional VLAN Models for Virtual Network Management

We are evaluating different alternatives for virtual network management. For example, Open vSwitch[1] is targeted at multi-server virtualization deployments, where the existing Linux networking stack presents some limitations. These environments are often characterized by highly dynamic end-points, the maintenance of logical abstractions, and (sometimes) integration with or offloading to special purpose switching hardware. Additionally, we are adding support to host-managed 802.1Q VLANs. In this case the OpenNebula network manager creates bridges

[1] http://openvswitch.org

and tagged interfaces for VMs as needed, when the VM is booted; and remove them upon VM disposal.

5.3.3 Automatic Setup of Simple TCP/UDP Firewall Rules for VMs

Each NIC defined in a VM can include simple filter rules based on destination ports. In particular the following can be used:

- `BLACK_TCP_PORTS`, `WHITE_TCP_PORTS`: To filter a set of specific ports or alternatively to only allow traffic to the set (individual ports or ranges) of ports specified.
- `BLACK_UDP_PORTS`, `WHITE_UDP_PORTS`: Same as above for the UDP protocol

This is implemented by dynamically creating iptables rules that captures the FORWARDING packets and setting up custom iptables chains for the VM interfaces.

5.3.4 VM Contextualization Using Virtual Network Information

Using `CONTEXT` parameter in the VM description file, it is possible to tell OpenNebula to create a contextualization image and to fill it with values. Variables inside `CONTEXT` section will be added to `context.sh` file inside the contextualization image. A new method has been added to specify these variables using variables from the Virtual Network template:

- `$NETWORK[<vnet_attribute>, NETWORK_ID=<vnet_id>]`: Any single value variable in the Virtual Network template, like for example:

  ```
  dns = "$NETWORK[DNS, NETWORK_ID=3]"
  ```

5.4 Others

5.4.1 Improved Fault Tolerance

Fault tolerance has been improved in OpenNebula 2.2 to automatically trigger recovery actions when a physical host or VM fails. Most of the actions are specified as hooks, which are custom scripts triggered by OpenNebula when a change in the state of a particular resource (Host or VM) is detected.

Failures are categorized depending on whether they come from the physical infrastructure (Host failures), from the virtualized infrastructure (VM failures) or from the virtual infrastructure manager (OpenNebula crash):

- Host failures: When OpenNebula detects that a host is down, a hook can be triggered to deal with the situation. OpenNebula comes with a script out-of-the-box that can act as a hook to be triggered when a host enters the `ERROR` state. This can be very useful to limit the downtime of a service due to a hardware failure, since it can redeploy the VMs on another host.

- VM failures: The VM lifecycle management can fail in several points. The following two cases should cover them:
  - VM fails: This may be due to a network error that prevents the image to be staged into the node, a hypervisor related issue, a migration problem, etc. The common symptom is that the VM enters the `FAILED` state. In order to deal with these errors, a VM hook can be set to resubmit the failed VM (or, depending the production scenario, delete it).
  - VM crash: This point is concerned with crashes that can happen to a VM after it has been successfully booted (note that here boot doesn't refer to the actual VM boot process, but to the OpenNebula boot process, that comprises staging and hypervisor deployment). OpenNebula is able to detect such crashes, and report it as the VM being in an `UNKNOWN` state. This failure can be recovered from using the `onevm restart` functionality.
- OpenNebula crash: OpenNebula can recover from a crash occurred in its core daemon, since all the information regarding infrastructure configuration and the state of the virtualized resources is stored on a persistent backend. Therefore, the `oned` daemon can be restarted after a crash, and all the running VMs will be reconnected with and monitored from this point onwards. Pending machines will be placed on a suitable host just as before the OpenNebula crash, as well as other non-transient states. However VMs not in a final state may need to be recovered manually, as in general the VM drivers are stateless.

5.4.2 Grouping of Physical Hosts in Clusters

OpenNebula 2.0 introduced support to cluster physical hosts. By default, all hosts belong to the `default` cluster. The administrator can create and delete clusters, and add and remove hosts from these clusters using the `onecluster` command. Thanks to this feature, the administrator can logically group hosts by any attribute like the service provided (e.g. `CLUSTER = production`), the physical location (e.g. `CLUSTER = roomA`) or a given characteristic (e.g. `CLUSTER = intel`).

6 Cloud like-Interfaces Specific for the Scientific Community

The Task 6.3 Cloud-like Interfaces Specific for the Scientific Community will define the cloud interfaces for the system to provide a Grid as a Service. To provide such a grid or Cluster as a Service interface the following extensions to current cloud approaches has to be considered: i) the ability to specify a grid service as a whole, ii) the ability to specify the characteristics of each component of a grid service including virtual networks and storage, iii) the ability to specify context information (e.g. CA certificates or component role) to each service component so it can be integrated within other grids and the integration of Grid services within the IaaS interface.

6.1 Cloud IaaS API

StratusLab has to complement existing grid services by exposing cloud-like APIs to users of the grid infrastructure. This will allow existing users to experiment with these new APIs and to develop new ways to use grid resources. StratusLab works towards the use of cloud-like Application Programming Interfaces (APIs) for managing cloud computing capabilities including resource sharing. That is, grid service users can use programmatic APIs to access to the shared resources in order to manage them. Thus, the service providers (the cloud/grid client entity) requests resources from the infrastructure providers or IT vendors to deploy the services and virtual machines.

Considering the abstraction layers which are included in StratusLab, two kinds of APIs should be considered:

- **The Service Manager Interface** (SMI) is the API for the service manager and the access point for service providers. From the state of the art in the D6.1 [12], TCloud API was selected as SMI.
- **Virtual Manager Interface** (VMI) is the API for accessing to the virtual machine manager, (OpenNebula for StratusLab). It hides the inherent underlying heterogeneity existing in cloud infrastructure providers from the service manager. From the state of the art in the D6.1 [12], OCCI API was selected as the SMI.

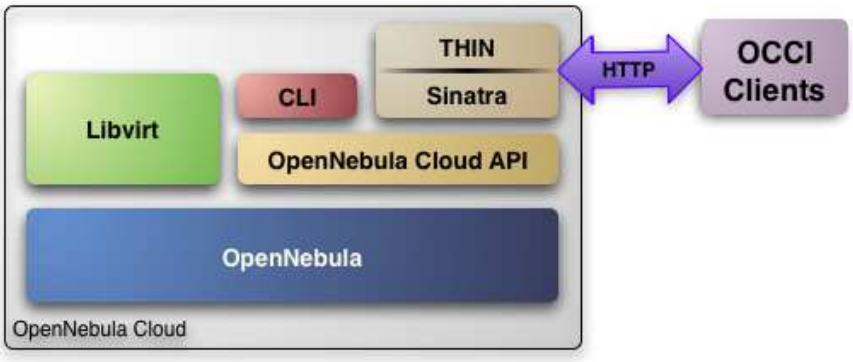

Figure 6.1: *OCCI implementation in OpenNebula.*

6.1.1 TCloud as the Claudia API

The TCloud API [14] is a RESTful, resource-oriented API accessed via HTTP which uses XML-based representations for information interchange. It constitutes an extension of some of the main standardization initiatives in Cloud management, such as the Open Virtualization Format (OVF), defined by the DMTF, and the vCloud specification [4], published by VMware and submitted to the DMTF for consideration. TCloud API defines a set of operations to perform actions over: i) Virtual Appliances (VApp), which is a Virtual Machine running on top of a hypervisor, ii) Hw resources the virtual hardware resources that the VApp contains, iii) Network both public and private networks, and iv) Virtual Data Center (VDC) as a set of virtual resources (e.g. networks, computing capacities) which incarnate VApps. TCloud API defines operations to perform actions over above resources categorized as follows: Self-Provisioning operations to instantiating VApps and VDC resources and Self-Management to manage the instantiated VApps (power on a VApp). In addition, it provides extensions on monitoring, storage, and so on.

StratusLab incorporate an implementation of the TCloud specification, the tcloud-server as Claudia API.

6.1.2 Enhancements in OGF OCCI Implementation in OpenNebula

The OpenNebula OCCI API is a RESTful service to create, control and monitor cloud resources based on the OGF OCCI 1.0 API specification. The OpenNebula OCCI service, as shown in Figure 6.1, is implemented upon the new OpenNebula Cloud API (OCA) layer that exposes the full capabilities of an OpenNebula private cloud; and Sinatra[1], a widely used light web framework.

The OCCI interface has been enhanced to expose more OpenNebula function-

[1] www.sinatrarb.com

ality for image management, including `save_as` operation and metadata attributes like `public` or `persistent`, for virtual network management, including metadata attributes, as well as for fine grain resource specification.

6.2 Integration of Grid services

6.2.1 Authentication in OpenNebula Based on LDAP and Grid/VOMS Certificates

The authentication and authorization mechanisms are separated into two services. An authentication proxy (developed in WP2) identifies users and passes the user information to the virtual machine manager. OpenNebula, through its "auth" module, will then make authorization decisions based on the specific request. OpenNebula will create new users dynamically, assigning them the default rights and quotas. Multiple mechanisms can be used simultaneously: username/password pairs maintained in a configuration file, username/password pairs from an LDAP server, grid certificates, and VOMS proxies created from grid certificates.

6.2.2 Authorization Based on Groups and Roles in OpenNebula

The authorization system in OpenNebula is being extended to support groups of users and access rules (roles) to manage OpenNebula resources. These groups and roles can be used, for example, to map attributes specified in VOMS certificates.

A group in OpenNebula is a way to categorize users, in the same fashion as UNIX groups. Therefore, a user is a member of a primary group and can also be in more than one secondary groups. There is a special administration group (similar to wheel) whose users can perform any action (by default).

The permissions are managed with an ACL, with a deny by default policy. Therefore, rules can be added to grant permissions, not to deny them or restrict other rules. If any of the rules allows the user to perform the requested action, it is granted, without having to establish a precedence.

There are also a set of hard-coded rules in the core, that cannot be modified and have preference over the ACL table. According to these hard-coded rules, the `oneadmin` user (ID=0) can perform any operation over any object and the owner of an object can perform any operation over that object.

7 Conclusions and Future Work

This document has provided an overview of the work done within the scope of WP6 in the first period of the project regarding the dynamic provision of grid services, the development of a scalable and elastic management of grid site and Cloud like-interfaces specific for the scientific community.

In this first period of the project, a grid site has been deployed and scaled. The introduction of the Claudia in the StratusLab distribution has allows to define the grid site as a whole, so that, the grid administrator only has had to specify the grid site features in a XML format (OVF) and Claudia has processed it and requests to the different involved components. In addition, through OVF and OpenNebula contextualization and some developed scripts, the different VMs involved in the grid site have been configured correctly.

The number of WNs in the grid site has to scale up and down according to the number of existing jobs and a number of grid administrator defined rules and thresholds. Claudia is in charge of providing this scalability functionality, by the existence of a rule engine which evaluates the defined rules. In the OVF, also it is possible to define the KPI used in this case the queue job utilization and rule which this KPI drives.

In addition some work has been done in order to adapt OpenNebula to the typical operations of a grid site. In particular, some virtual resource placement heuristics are being evaluated to optimize different infrastructure metrics. Some work is being done towards cloud-aware image management techniques by the development of a image repository in OpenNebula and support of multiple storage backbends. Finally, some work in techniques for the management of cloud-aware networks has been carried out.

The task related to cloud-like interfaces has started by the introduction of Cloud service API like TCloud and OCCI which has been discussed and explained in previous documents.

As future work, the work on grid site scalability is going to continue in order to satisfy the requirements taken from experimentation. This means to add the possibility to expand and contract a grid site according to the job queue utilization. Currently, Claudia only scale up or down a concrete VM, but the idea is to be able to define the way to scale up and down. Also considering in scalability, the VMs un-deployed for scaling down, have to be decided by the CE according to its usage. Finally, some process will be to implement a “lazy scale-down” strategy according

to which the scale-down pace is lower than the scale-up counterpart in order to better accommodate job fluctuations and speedup job execution be reducing the time required for site scale-up.

Equally important is the issue of resource accounting and the integration of grid sites accounting system with the respective cloud services accounting. This issue has been identified from the early stages of the project but work on it had to be postponed until we gained enough experience from the operation of cloud and grid services in the context of WP5. This experience will help us identify an integration path between the two worlds and design a solution for grid/cloud interoperation. In this process it will be also necessary to include relevant stakeholders from the domain of grid operations (e.g. through the collaboration with EGI-InSPIRE project) and middleware development (EMI).

Regarding the scalable and elastic management of grid site infrastructure, future work will be focused on the use of different storage backends to access external image catalogs, thus integrating the OpenNebula Image Repository with the StratusLab Marketplace as well as third-party providers like Amazon S3. Support for new VLAN models, based on Open vSwitch and host-managed 802.1Q VLANs, will be also implemented.

Finally, in order to provide an interface for grid or cluster as a service, some work is going to be done to extend current cloud approaches. For example, the ability to specify context information for each service component (e.g. CA certificates or component role) so it can be integrated within other grids; or the integration of more grid services within the IaaS interface.

Glossary

ACL	Access Control List
Appliance	Virtual machine containing preconfigured software or services
APEL	Accounting Processor for Event Logs
Appliance Repository	Repository of existing appliances
CDDLM	Configuration Description, Deployment, and Lifecycle Management
CE	Compute Element
DHCP	Dynamic Host Configuration Protocol
DMTF	Distributed Management Task Force
Front-End	OpenNebula server machine, which hosts the VM manager
Hybrid Cloud	Cloud infrastructure that federates resources between organizations
IaaS	Infrastructure as a Service
IP	Infrastructure Provider
Instance	a deployed Virtual Machine
JRA	Joint Research Activity
KPI	Key Performance Indicator
Machine Image	Virtual machine file and metadata providing the source for Virtual Images or Instances
NFS	Network File System
Node	Physical host on which VMs are instantiated
OASIS	Organization for the Advancement of Structured Information Standards
OCCI	Open Cloud Computing Initiative
OGF	Open Grid Forum
OVF	Open Virtualization Format
Public Cloud	Cloud infrastructure accessible to people outside of the provider's organization
Private Cloud	Cloud infrastructure accessible only to the provider's users
Regression	Features previously working which breaks in a new release of the software containing this feature
Service Manager/SM	A toolkit to provides Service Providers to dynamically control the Service provisioning and scalability
Service Provider/SP	The provider who offers the application to be deploy in the Cloud
SMI	Service Manager Interface

SLA	Service Level Agreement
SE	Storage Element
SSD	Solution Deployment Descriptor
Virtual Machine / VM	Running and virtualized operating system
TCloud	It is a RESTful API use for Cloud service management
VMI	Virtual Manager Interface
VIM	Virtual Infrastructure Manager
VO	Virtual Organization
VOMS	Virtual Organization Membership Service
Web Monitor	Web application providing basic monitoring of a single StratusLab installation
Worker Node	Grid node on which jobs are executed

References

[1] J. Cáceres, L. M. Vaquero, L. Rodero-Merino, A. Polo, and J. J. Hierro. Service Scalability over the Cloud. In B. Furht and A. Escalante, editors, *Handbook of Cloud Computing*, pages 357–377. Springer US, 2010. 10.1007/978-1-4419-6524-0_15.

[2] C. Chapman, W. Emmerich, F. G. Márquez, S. Clayman, and A. Galis. Software architecture definition for on-demand cloud provisioning. In *HPDC '10: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 61–72, New York, NY, USA, 2010. ACM.

[3] Cluster Resources. Terascale Open-Source Resource and QUEue Manager. http://www.clusterresources.com/products/torque-resource-manager.php.

[4] DMTF. Open virtualization format specification. Specification DSP0243 v1.0.0d. Technical report, Distributed Management Task Force, Sep 2008. https://www.coin-or.org/OS/publications/optimizationServicesFramework2008.pdf.

[5] E. Huedo, R. Moreno-Vozmediano, R. Montero, and I. Llorente. Architectures for Enhancing Grid Infrastructures with Cloud Computing. In M. Cafaro and G. Aloisio, editors, *Grids, Clouds and Virtualization*, Computer Communications and Networks, chapter 3, pages 55–69. Springer, 2011.

[6] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. T. Foster. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, 13(5):14–22, 2009.

[7] StratusLab. APEL Image. Online resource. http://appliances.stratuslab.eu/images/grid/apel/sl-5.5-x86_64-grid.apel/1.0/sl-5.5-x86_64-grid.apel-1.0.img.gz.

[8] StratusLab. CE Image. Online resource. http://appliances.stratuslab.eu/images/grid/ce/sl-5.5-x86_64-grid.ce/2.0/sl-5.5-x86_64-grid.ce-1.0.img.gz.

[9] StratusLab. SE Image. Online resource. http://appliances.stratuslab.eu/images/grid/se/sl-5.5-x86_64-grid.se/2.0/sl-5.5-x86_64-grid.se-2.0.img.gz.

[10] StratusLab. WN Image. Online resource. http://appliances.stratuslab.eu/images/grid/wn/sl-5.5-x86_64-grid.wn/2.0/sl-5.5-x86_64-grid.wn-1.0.img.gz.

[11] Stratuslab Consortium. Deliverable 4.1 Reference Architecture for StratusLab Toolkit 1.0. Online resource., 2010. http://stratuslab.eu/lib/exe/fetch.php?media=documents:stratuslab-d4.1-v1.0.pdf.

[12] Stratuslab Consortium. Deliverable 6.1 Cloud-like Management of Grid Sites 1.0 Design Report. Online resource., 2010. http://stratuslab.eu/lib/exe/fetch.php/documents:stratuslab-d6.1-v1.0.pdf.

[13] Stratuslab Consortium. Deliverable 6.2 Cloud-like Management of Grid Sites 1.0 Software. Online resource., 2011. http://stratuslab.eu/lib/exe/fetch.php/documents:stratuslab-d6.2-v1.1.pdf.

[14] Telefónica. TCloud API Specification, Version 0.9.0. Online resource., 2010. http://www.tid.es/files/doc/apis/TCloud_API_Spec_v0.9.pdf.

[15] TID. The Claudia project. Online resource., 2010. http://claudia.morfeo-project.org/wiki/index.php/Main_Page.

[16] YAIM. YAIM Ain't an Installation Manager. http://yaim.info.