Arquitecturas para la Federación de Proveedores Cloud

Daniel Molina Aranda

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTESNE DE MADRID

Trabajo Fin Máster en Ingeniería de Computadores

2010-2011

Director/es y/o colaborador:

Rubén Santiago Montero
Ignacio Martín Llorente

Convocatoria: Septiembre 2011
Calificación: 8

Autorización de difusión

Daniel Molina Aranda

2010-2011

El/la abajo firmante, matriculado/a en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Arquitecturas para la Federación de Proveedores Cloud", realizado durante el curso académico 2010-2011 bajo la dirección de Rubén Santiago Montero e Ignacio Martín Llorente en el Departamento de Arquitectura de Computadores, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen en castellano

Este trabajo analiza el modelo de cloud híbrido, un paradigma que combina los depliegues de cloud privado con los recursos ofrecidos por cloud públicos. Este nuevo modelo no está totalmente desarrollado todavía, y hay que dedicar mucha más investigación y desarrollo antes de conseguir que despliegues multi-cloud puedan ser usados en producción. En este trabajo se realiza un estudio de las limitaciones y desafíos de este modelo y a su vez se plantean algunas de la técnicas más comunes para lidiar con estos puntos. También se presentará una arquitectura híbrida basada en la herramienta OpenNebula, intentando superar algunos de los desafíos y presentar experiencias reales con esta arquitectura y Amazon EC2

Palabras clave

cloud computing, hybrid cloud, federación, interoperabilidad

Abstract

This work analyzes the Hybrid Cloud computing model, a paradigm that combines on-premise private clouds with the resources of public clouds. This new model is not yet fully developed, and there are still a lot of work to be done before true multi-cloud installations become mature enough to be used in production environments. A review of some of its limitations and the challenges that have to be faced is done in this work, and we also include some common techniques to address the challenges studied. We will also present a hybrid cloud architecture based on the OpenNebula Cloud toolkit, trying to overcome some of the challenges, and present some real-life experiences with this proposed architecture and Amazon EC2.

Keywords

cloud computing, hybrid cloud, federation, interoperability

Índice general

Índice I

1. Introduction 1

2. Challenges for Hybrid Cloud Computing 4
- 2.1. Cloud Interfaces . . . 4
- 2.2. Networking . . . 6
- 2.3. Heterogeneous Resources . . . 7
- 2.4. Master Images Management . . . 8
- 2.5. Virtual Image Formats . . . 12
- 2.6. Virtual Machine Contextualization . . . 14
- 2.7. Non-techinical Challenges . . . 17
  - 2.7.1. Multiple Pricing Models . . . 17
  - 2.7.2. Legal Restraints . . . 17

3. Proposed Solutions 18
- 3.1. Hybrid Cloud, Compute and Network . . . 18
  - 3.1.1. Sun Grid Engine Use Case . . . 20
  - 3.1.2. Nginx Use Case . . . 22
- 3.2. Hybrid Storage . . . 25
  - 3.2.1. EC2 AMIs Use Case . . . 30

4. Conclusion 35

Bibliography 38

A. Step by Step Service Deployment in OpenNebula 39
- A.1. Preparing the Network . . . 39
- A.2. Modifying the template . . . 40
- A.3. Running the VM . . . 40
- A.4. Running the VM again with CONTEXT . . . 41

B. Hybrid Cloud Deployment powered by OpenNebula 42
- B.1. EC2 Configuration . . . 43
- B.2. Configuration . . . 43
  - B.2.1. OpenNebula Configuration . . . 43
  - B.2.2. Driver Configuration . . . 43
- B.3. EC2 Specific Template Attributes . . . 44

B.4. Usage 44

Capítulo 1

Introduction

Nowadays, Infrastructure as a Service (IaaS) clouds are considered a viable solution for the on-demand provisioning of computational resources. Since its popularization in 2006 by the Amazon Elastic Computing Cloud (EC2) [1], the IaaS paradigm is being adopted by many organizations not only to lease resources from a Cloud provider, but also to implement on-premise IaaS Clouds. The former, and original usage, is usually referred as Public Clouds while the latter is commonly named Private Cloud. [10] [8]

Public as well as private clouds have rapidly evolved since the advent of the IaaS cloud paradigm. The public IaaS market has been enriched with multiple providers each one with different price model and offers, Cloud interfaces and APIs and, even disparate, features. The private ecosystem is not different and multiple technologies both open-source and private can be used today to build on-premise clouds. Again the features, characteristics and adoption levels greatly vary among these technologies [6].

Private clouds were designed to address specific needs that are missing in a Public cloud, namely:

- Security, sensible data may not be stored nor processed in an external resource
- Legal constraints, the laws of most countries impose several limitations on the geographical location of digital data

[1] http://aws.amazon.com/ec2/

- On-premise infrastructures, most organizations rely on their own resources to address the most important computational needs.

Although there are reasons for Private Clouds to exist, they present some of the traditional problems associated with running a data-center that Public Clouds try to mitigate, notably adjusting its dimension so the utilization is maximized while satisfying a given demand.

The solution to this problem is usually termed as Hybrid Cloud computing. A Hybrid Cloud is a special case of Cloud federation where instead of connecting a private cloud with a partner infrastructure running another interoperable private IaaS cloud, we connect a public one providing infiniteçapacity for deploying virtual machines on demand. We use the Private Cloud, to satisfy the average or sensible demands, and the Public Cloud, to outsource resources for low security or peak demands.

In the current cloud computing scenario we can find some utilities such as Deltacloud[2] or Libcloud[3] that allow us to manage different clouds at the same time but in a separate way. What we are trying to expose in this work is how we can interconnect our infrastructure with external resources and use them as if they were local.

In this work we will analyze the Hybrid Cloud computing model, starting with a review of some of its limitations and some techniques to overcome these problems. We will also include some real-life experiences with this model using the OpenNebula Cloud toolkit and Amazon EC2.

The first part of the proposed solutions will include an analysis of the previous work done in the hybrid cloud scenario exposed in two papers: "Multicloud deployment of computing clusters for loosely coupled MTC applications" [5], a deployment of a Sun Grid Engine computing cluster and "Elastic management of cluster-based services in the cloud" [6], a implementation of a virtualized web cluster that apply some of the proposed techniques to deal with the compute and network challenges.

[2] http://incubator.apache.org/deltacloud/
[3] http://libcloud.apache.org/

The second part of this work is focused in the hybrid storage. We propose an architecture to provide OpenNebula with a interoperable storage environment that will mitigate some of the limitations that are exposed at the beginning of this work.

Capítulo 2

Challenges for Hybrid Cloud Computing

Although Hybrid Cloud computing promises the best of the Public and Private Clouds, it presents some practical difficulties that limit its application to real-life workloads, or prevent an optimal combination for a given use-case [5]. This section analyzes these challenges and reviews some common techniques to address them.

2.1. Cloud Interfaces

Cloud interoperability is probably one of the aspects that is receiving more attention by the community. The need for interoperable clouds is two folded: first, the ability to easily move a virtualized infrastructure among different providers would prevent vendor locking; and secondly, the simultaneous use of multiple clouds –geographically distributed– can also improve the cost-effectiveness, high availability or efficiency of the virtualized service.

The cloud is often pictured as an infinite pool of computational resources that are always "there". But recent events like the EC2 outage of April 2011 have demonstrated that moving services to the cloud computing paradigm is not exempt from the possibility of service downtime. This is one of the reasons why interoperability is such a big concern: to achieve replication and high availability across different cloud providers. But the current Public Cloud ecosystem is far from being homogeneous and each provider exposes its own Cloud interface. Moreover the semantics of these interfaces are adapted to the particular services that each provider offers to its costumers. This means that apart from creating virtual

machines, each provider has a different definition of what a storage volume is, or offers services not available in other providers, like firewall services, additional storage or specific binding to a custom IP.

Among the most known and used cloud providers are Amazon Elastic Compute Cloud, Rackspace[1] and ElasticHosts[2]. These are just three examples that offer completely different APIs, pricing models and underlying infrastructure; and of course there are many more providers to choose from like FlexiScale [3], Terremark[4], GoGrid[5], etc.

These challenges have been identified by the cloud community, see for example the Open cloud manifesto.org, and several standardization efforts are working in different aspects of a Cloud, from formats, like the Open Virtualization Format specification, 2010, from the distributed management task force (DMTF), to cloud interfaces, for example the "Open cloud computing interface"[6], 2011 from the Open Grid Forum . In this area, there are other initiatives that try to abstract the special features of current implementations by providing a common API to interface multiple clouds, see for example the Deltacloud project or Libcloud.

A traditional technique to achieve the inter-operation in this scenario, i.e. between middleware stacks with different interfaces, is the use of adapters. In this context, an adapter is a software component that allows one system to connect to and work with a given cloud.

There are several open source and commercial virtual infrastructure managers that can manage a pool of physical resources to build an Infrastructure as a Service (IaaS) private cloud. Some examples are VMware vCloud Director[7], OpenStack[8] or Eucalyptus[9].

Some of these virtual infrastructure managers can even manage both local on-premise physical resources and resources from remote cloud providers. The remote provider could be

[1] http://www.rackspace.com/
[3] http://www.flexiant.com/products/flexiscale/
[4] http://www.terremark.es/
[5] http://www.gogrid.com/
[6] http://occi-wg.org/
[7] http://www.vmware.com/products/vcloud-director/overview.html
[8] http://www.openstack.org/
[9] http://open.eucalyptus.com/

a commercial Cloud service, such as Amazon EC2 or ElasticHosts, or a partner infrastructure running another interoperable private IaaS cloud. Some examples of this technology are OpenNebula[10] or Nimbus[11].

In the following sections we will describe a hybrid architecture based on the OpenNebula toolkit, an open-source virtual infrastructure manager that can manage remote cloud provider resources using different pluggable drivers; presenting those remote clouds seamlessly as if they were just another virtualization host of the local infrastructure. Users continue using the same private and public Cloud interfaces, so the federation is not performed at service or application level but at infrastructure level by OpenNebula; a remote cloud provider is managed as any other OpenNebula host that may provide "infinite" capacity for the execution of virtual machines.

2.2. Networking

Virtual Machines running in the Private and Public Cloud are located in different networks, and may use different addressing schemes, like public addresses, or private addresses with NAT. However, usually it is required that some of them follow a uniform IP address scheme (for example, to be located on the same local network), so it could be necessary to build some kind of overlay network on top of the physical network to communicate these Virtual Machines. There is also an outstanding security concern [7], since the services running on the remote cloud provider's resources could exchange strategic business data, or sensitive customer's information.

In this context, there are some interesting research proposals like ViNe [11], CLON [4], etc., or some commercial tools, like VPN-Cubed[12], which provide different overlay network solutions for grid and cloud computing environments. This network overlay provides the virtual machines with the same address space by placing specialized routers in each cloud

[10] http://www.opennebula.org/
[11] http://www.nimbusproject.org/
[12] http://www.cohesiveft.com/vpncubed/

–usually user-level virtual routers– that act as gateways for the virtual machines running in that cloud. Note that these solutions do not require to modify the service virtual machines.

As one would expect, some cloud providers offer extra services to protect or isolate the network connecting the customer's virtual machines. For instance, Amazon EC2 has a service called Amazon Virtual Private Cloud[13] that lets customers provision a virtual network, private and isolated from the rest of their cloud. This service is complemented by the option to create a Hardware Virtual Private Network (VPN) connection between the on-premise infrastructure and the Virtual Private Cloud, effectively creating the complete network scenario for hybrid deployments of services.

But again, this is a provider's specific solution, not interoperable with the similar services offered by other providers, if they are offered at all. This leaves out the possibility to deploy a service in several public clouds. For example, in ElasticHosts there is a private network VLANs service, but external components have to access the cloud's virtual network through a VPN software running in the virtual machines.

The most flexible and provider-independent solution is to install and configure this overlay network inside the virtual machines, using for example OpenVPN [14], a cross-platform open source software. This approach requires each virtual machine image to be configured individually.

2.3. Heterogeneous Resources

Usually each Cloud provider offers different virtual machine instance types that differ in their processing and storing capacity. The combination of these different virtual machines with those obtained from the Private cloud may lead to workload unbalance or a wrong public-to-private virtual machine distribution.

Just comparing three of the main providers, EC2, ElasticHosts and Rackspace, we can see important infrastructure differences: Amazon EC2 virtualization is based on Xen, and the

[13] http://aws.amazon.com/vpc/
[14] http://openvpn.net/

instances have data persistence only if they boot from an Amazon Elastic Block Storage[15]. The virtual machine capacity can be chosen from a list of predefined instance types with different virtual hardware configurations.

The ElasticHosts infrastructure is based on KVM, all the storage provided is persistent; and there is no notion of instance type as users can customize the hardware setup of the virtual machines and define arbitrary capacity for them.

Rackspace virtualization infrastructure is based on Xen and XenServer, and as with ElasticHosts the storage is persistent. There is also a list of predefined instance type with variable amount of RAM and disk space; but unlike in EC2, there is no choice of the number of virtual processor cores.

2.4. Master Images Management

In general, a virtualized service consists in one or more components each one supported by one or more virtual machines. Instances of the same component are usually obtained by cloning a master image for that component, that contains a basic OS installation and the specific software required by the service.

In a hybrid environment, the master images for the virtualized services must be available in the local infrastructure and each one of the remote cloud providers. This presents a complex challenge, as cloud providers use different formats and bundling methods to store these master images; and some providers allow users to upload their own image files, whereas with others only a predefined set of pristine OS installations are available as a base for customization.

There are two main strategies to prepare these images: create and customize an image manually for each one of the providers; or craft them once in the local infrastructure, and then upload the image files to each provider.

We will review later the specific challenges associated with the task of converting and up-

[15] http://aws.amazon.com/ebs/

loading existing image files to the cloud providers; but to summarize the main complications are the lack of a common API and a common image file format. To avoid these complications, the first approach taken in the first hybrid setup implementations [6] [5] is to assume that suitable service component images have been previously packed and registered in each cloud provider storage service manually. So when a virtual machine is to be deployed in a remote cloud the virtual infrastructure manager can skip any image conversion or transfer operation.

Note that this approach minimizes the service deployment time as no additional transfers are needed to instantiate a new service component. However there are some drawbacks associated to the storage of one master image in each cloud provider: higher service development cycles as images have to be prepared and debug for each cloud; higher costs as clouds usually charge for the storage used; and higher maintenance costs as new images have to be distributed to each cloud.

To perform this task of reproducing the same environment in the local and remote providers images, different system automation tools are available. A basic OS installation for the most common Linux distributions can be deployed in any of the existing cloud providers, either because the provider allows users to upload any image file, or because they offer a list of images to select from. There are commercial and open-source system automation tools that use a series of scripts to install and configure a system starting from that clean OS setup; such as Puppet [16], Chef [17] or CFEngine [18].

These automation tools require the remote instance to be running, and to configure the connection and credential details. Then they log into that virtual machine to install and configure the desired software. Once the process is complete, that virtual machine should be saved back to the provider's storage; to be instantiated as many times as needed by the local virtual infrastructure manager.

[16] http://www.puppetlabs.com/
[17] http://www.opscode.com/chef/
[18] http://cfengine.com/

Another possibility not yet explored by any virtual infrastructure manager product would be to integrate the system automation tool into the virtual infrastructure manager. Each virtual machine instance would start always as a base OS installation, and configured by the system automation tool right after the deployment. This approach offers some advantages and drawbacks. The clear advantage is that the administrator of the virtualized service can update the configuration scripts, and have those changes available to all the new virtual machine instances without further effort. On the other hand, any administrator knows how to log in and manually configure a virtual machine; and this tight coupling of the virtual infrastructure manager and the system automation tool presents a new configuration mechanism that can be unknown and time consuming to learn. It also imposes an extra service deployment time, as each instance has to wait until the installation and configuration is complete.

The previous approach requires the user of the hybrid cloud architecture -that is, the administrator or developer of the service to virtualize- to be aware of the available remote providers and perform a great amount of manual configuration and testing in order to be able to take advantage of the hybrid setup. The next step for this kind of infrastructures is to free the users of this task, and let the virtual infrastructure manager make the necessary operations to deploy a registered virtual machine in a remote provider with the same zero configuration needed to deploy it in the local hypervisors.

This of course presents new challenges. The main obstacle is the fact that, at the time of this writing, only a few public cloud providers allow users to upload any generic image and instantiate virtual machines from it. Amazon EC2 provides a number of tools to create an Amazon Machine Image (AMI) from an existing system; once the custom AMI is created, it can be "bundled", using the EC2 API or the Web Management Console. ElasticHosts lets users upload images in raw format using their API, a ftp server, their web interface or even sending them an usb hard drive.

Each one of the above providers exposes a different API for uploading the images to

their environments. Moreover, some of these providers add a custom method for bundling the images to upload them in smaller parts.

Assuming the problem of how to upload image files to different cloud providers is solved, there is still another issue to take into consideration. The images can be uploaded to the remote infrastructures right after the user registers the new image in the local virtual infrastructure manager storage, or on-demand, only when a virtual machine is instanced in the cloud provider.

Uploading the images only when needed can be convenient from an economical point of view, since the providers charge for the storage services, but it can take a long time and will definitely affect the deployment time for new virtual machines. The other option would be to keep an updated copy of the Virtual Machine image in each one of the available cloud providers beforehand. This way the deployment times will be drastically reduced, but if the images contain non-static data then it is required to implement a mechanism to keep the remote copies updated to the latest version.

The Image files transfer can occur also in the other way, i.e. downloading existing Images from a remote cloud provider to later deploy them in the local infrastructure. This can be interesting for organizations that already have services running in a cloud provider, and want to migrate those instances to their private cloud.

This feature is meaningful by its own, even if it is implemented without the possibility to upload local images to the cloud providers. In Amazon EC2 there is a great number of AMIs ready to use shared by the community, containing lots of different installations and configurations, so this feature would allow local infrastructure users access to a rich Image catalog.

Importing those AMIs to the local infrastructure is not an easy task. The API interoperability is again the main issue, as well as the different bundling and Image formats used by the providers. As we will see later, the images shared in EC2 expect the Amazon EC2 contextualization metadata server, which has to be replicated for the imported machines to

work identically in the local virtualization infrastructure as in EC2.

At the end of this chapter we propose a new architecture for OpenNebula that will provide this functionality using adaptors, a different set of drivers for each provider.

2.5. Virtual Image Formats

We have discussed the problems associated with the existing methods to upload virtual machine images to the cloud providers; or download Images in a remote catalog to be registered in the local infrastructure. However, the problem is not only how to move files, but what kind of files have to be transferred. There are different kinds of image formats to represent a Virtual Machine hard disk, and even depending on the hypervisor it is required to provide an image file containing a whole disk or a file for each separate partition that are mounted in separated logical volumes.

Each cloud provider supports different formats depending on their internal architecture and the hypervisors they use. We have compiled a list of the most common formats used by current hypervisors and cloud providers:

- RAW

  A raw image file has no specific format, it only contains the raw disk data.

- VMDK

  The Virtual Machine Disk (VMDK) file format is a type of virtual appliance developed for VMware products.

  The VMDK Image Format Specification is available to third parties, so it is natively supported by other products like QEMU, SUSE Studio and VirtualBox. Also QEMU provides the qemu-img utility to convert VMDK images to different formats.

- VHD

A Virtual Hard Disk (VHD) is a virtual hard disk file format initially used only by Microsoft Virtual PC. Later Microsoft used it in Hiper-V, a hipervisor-based virtualization technology.

There are also third-party products with VHD support: VirtualBox, VMware ESX Server, WMware Workstation and Citrix XenServer natively support VHD format.

- VDI

  Virtual Desktop Image (VDI) is the name of the default storage format for VirtualBox containers. VirtualBox provides a command-line utility, VBoxManage, that transforms images from VDI to different formats, including VMDK, VHD and RAW.

- QCOW

  The QEMU Copy On Write (QCOW) image format is one of the disk image formats supported by the QEMU processor emulator. It is a representation of a fixed size block device in a file that delays allocation of storage until it is actually needed. The benefits that it offers in its latest version (QCOW2) over using raw dump representation include:

  1. Smaller file size, even on filesystems which don't support holes (i.e. sparse files)
  2. Copy-on-write support, where the image only represents changes made to an underlying disk image
  3. Snapshot support, where the image can contain multiple snapshots of the images history
  4. Optional zlib based compression
  5. Optional AES encryption

Because of the low level differences between all the above described image formats, the only solution to achieve interoperability is transform the image files for the target hypervisor

or cloud provider. There is a wide choice of applications to carry out this task, some of them reviewed in the previous list of formats. In the virtual infrastructure manager scenario, this means that before uploading an image to a cloud provider, an extra step is needed to transform the image format.

There is another issue related to the way images are formatted and stored. A Linux virtual machine requires to boot an initial ramdisk (initrd) and kernel. They can be provided as files separated from the main image or they can be included in the disk.

In a Xen environment, the virtual machines do not usually have bootable disks; instead, a Xen kernel stored in a different file is used to boot the virtual machine, or DomU in Xen terminology. KVM however typically allows guest virtual machines to run unmodified kernels, and as a result usually are defined with disks and partitions including a boot loader such as grub installed in the master boot record (MBR).

An approach to solve this difference and be able to manage Xen images in a similar way as the KVM ones is to use a special bootloader. PyGrub enables Xen guest machines to boot from a kernel inside the image filesystem, instead of the physical host filesystem. This makes also easier to update the kernels to newer versions from the virtualized operative system.

2.6. Virtual Machine Contextualization

One of the most important features to make use of the "install-once-deploy-many" approach is the ability to pass context data to the virtual machine at boot time. Even with a fully installed and configured virtual machine image, it is usually necessary to perform a minimal service contextualization each time the machine is deployed.

For instance, we could register into our hybrid setup infrastructure a virtual machine containing a slave execution node for a grid computing cluster, that needs to contact the head master virtual machine at boot to register itself into the cluster. Instead of hard-coding the head node's IP, the virtual infrastructure manager can provide that IP and a small set

of configuration files at boot. This allows the service users to deploy more than one grid cluster service with only one set of registered virtual machines.

The instance level contextualization occurs once per virtual machine instantiation and as such must be automated. Currently, there is no standard procedure to do this and each cloud provider uses a different method, and even the same provider can have different methods depending on the operative system running in the virtual machine.

Basic instance level contextualization enables access to data outside the instance. This data can be read by the virtual machine at boot time, and used in initialization or configuration scripts to contextualize the guest operating system. Most cloud providers use an ISO image or a Metadata Server to serve these data to the virtual machines.

The first option consists in mounting a CDROM device or attached disk in the virtual machine, containing dynamically created data provided by the user. The virtualized service can then access that data using standard operating system tools. This process is managed by the infrastructure manager, and it is the one used by OpenNebula. This mechanism is compliant with the OVF standard.

The method provided by OpenNebula to give configuration parameters to a newly started virtual machine instance is using an ISO image. This method is network agnostic so it can be used also to configure network interfaces. Users can specify in the virtual machine template the contents of the ISO file and the configuration parameters that will be written to a file for later use inside the virtual machine instance.

In Figure 2.1 we see a Virtual Machine with two associated disks. The Disk Image holds the filesystem where the Operating System will run from. The ISO image has the contextualization for that Virtual Machine Instance:

1. context.sh: file containing configuration variables, filled by OpenNebula with the parameters specified in the Virtual Machine Template.

2. init.sh: script called by the Virtual Machine instance at start that will configure specific services for this instance.

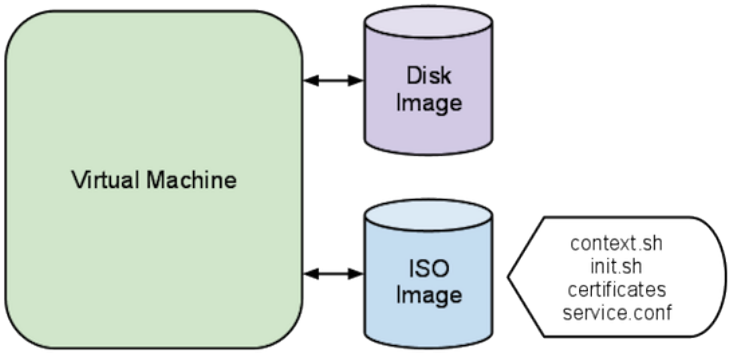

Figura 2.1: *OpenNebula contextualization*

3. certificates: sample directory that contains certificates for some service.

4. service.conf: sample file, containing service configuration.

An alternative method is to store the contextualization data in a metadata server, accessible from the virtual machine. This is the method used by some cloud providers such as EC2 or virtual infrastructure managers like Nimbus. The metadata server is configured to allow access only from virtual machine instance, that has to use a Query API to retrieve the data from a web server.

In a hybrid setup, this contextualization presents a problem since the images have to be prepared to use the local virtual infrastructure manager, and each one of the available cloud providers contextualization mechanisms. The basic approach to this challenge is to have different images in the local infrastructure and remote providers, adjusted to the specific operation.

A more convenient solution is to have the virtual infrastructure manager replicate the contextualization mechanisms of the cloud providers. At the end of this chapter we introduce a proposed architecture to add new functionality to OpenNebula that will enable it to import EC2 AMIs to the local image repository, and reproduce the metadata server to properly contextualize the instances.

2.7. Non-techinical Challenges

There are other several challenges in Hybrid Cloud Computing that are not technical difficulties. The deep analysis of this kind of challenges is out of the scope of this work; but for completeness we will introduce some of them in this section.

2.7.1. Multiple Pricing Models

Probably the main reason to outsource peak demands to a Public Cloud is to save costs by better dimensioning the private infrastructure. However, the disparate pricing models available in the Public Cloud market make it difficult to optimize the cost scheme for the hybrid setup, and usually requires advanced placement policies.

Not only the cloud providers have different prices, but they also use different billing schemes. For example, Amazon EC2 offers a service called Spot Instances, in which customers bid on unused Amazon EC2 capacity and run instances for as long as their bid exceeds the current Spot Price, that changes periodically. There is no equivalent in ElasticHosts or Rackspace.

2.7.2. Legal Restraints

Moving the computing resources to a hybrid cloud setup can be a difficult task if the virtualized services manage data restrained by legal implications, such as customers private data, or company information that needs to be tightly protected.

The physical location of the data is important since different jurisdictions mean the data might be secure in one provider but may not be secure in another. What's even more, the data may be restrained by government laws that forbid it to exit the country's territory.

The virtual infrastructure manager would need to implement a strong Service Level Agreement (SLA) mechanism to cope with all the legal restraints, so the company could define that some virtual machines are not to leave the local infrastructure, while others could be deployed but only under certain circumstances.

Capítulo 3

Proposed Solutions

3.1. Hybrid Cloud, Compute and Network

This section describes a basic hybrid cloud setup proposal based on OpenNebula, that addresses some of the previously reviewed challenges, namely: the use of adaptors to interface multiple Cloud interfaces and the use of virtualized routers to easily interconnect private and public virtual machines. In the following sections we show two use cases where the presented architecture was implemented and it performed successfully.

The first step to achieve a hybrid cloud setup is to install a virtual infrastructure manager in the local infrastructure so it can be abstracted as a IaaS private cloud; and then add capabilities so it can also interface with external cloud providers.

OpenNebula is an open-source virtual infrastructure manager able to build any type of IaaS cloud: private, public and hybrid, with unique characteristics for the integration of multiple technologies. It provides the functionality needed to deploy, monitor and control virtual machines on a pool of distributed physical resources, usually organized in a cluster-like architecture. The OpenNebula core orchestrates three different areas to effectively control the life-cycle of a virtual machine: virtualization, image management and networking.

OpenNebula exhibits a pluggable architecture so specific operations in each of the previous areas (e.g. shutdown a virtual machine or clone a disk image) are performed by specialized adapters. This way, the interoperability issue is managed entirely by the drivers,

so OpenNebula manages an external public cloud just as it was another local resource. Therefore, any virtualized service can transparently use the public cloud.

Local infrastructure can be supplemented with computing capacity from several external clouds to meet service requirements, to better serve user access requests, or to implement high availability strategies. In particular the integration of a new public cloud requires the development of two plug-ins: virtualization, to interface with the cloud provider and perform virtual machine operations; and information, to limit the capacity that the organization wants to put in the cloud provider.

An OpenNebula virtualization plug-in translates the high-level actions, like "deploy", "shutdown", etc. into specific hypervisor operations. In the case of the presented architecture, the plug-ins for remote cloud providers translate the OpenNebula core actions into Amazon EC2 or ElasticHosts API calls.

The information plug-in gathers information about the physical host and hypervisor, so the OpenNebula scheduler knows the available resources and can deploy the virtual machines accordingly. The Amazon EC2 and ElasticHosts plug-ins cannot provide much details about the physical infrastructure or the available resources, since they are offered to the user as "infinite". In this case, the plug-ins are used to report a fixed amount of resources (CPU, free memory, etc.) to limit how many virtual machines are deployed in the cloud provider. This prevents a service with a high workload to upscale without limit, which means an uncontrolled cost to the organization.

Having the API interoperability managed by the virtual infrastructure manager OpenNebula, the virtual machines still need a way to communicate with each other. The proposed solution is to use Virtual Private Network (VPN) technology to interconnect the different cloud resources with the in-house data center infrastructure in a secure way. In particular, we propose OpenVPN software to implement Ethernet tunnels between each individual virtual machine deployed in a remote cloud and the local infrastructure LAN.

In this setup, which follows a client-server approach, the remote cloud resources, con-

figured as VPN clients, establish an encrypted VPN tunnel with the in-house VPN server, so that each client enables a new network interface which is directly connected to the data center LAN. In this way, resources located at different clouds can communicate among them, and with local resources, as if they were located in the same logical network. This allows them to access common LAN services (NFS, NIS, etc.) in a transparent way, as local resources do.

It is obvious that the VPN software can introduce some extra latencies in the communication between the front-end and the remote back-end nodes, however, it also involves important benefits. First, although virtual machines deployed on Amazon EC2 have a public network interface, they can be configured to only accept connections through the private interface implemented by the OpenVPN tunnel; this configuration provides the same protection degree to the remote back-end nodes than to the local ones, since the front-end can apply the same filtering and firewalling rules to prevent them from unauthorized or malicious access. Second, from the point of view of the virtualized service all nodes (either local or remote) are accessed in a similar way through the private local area network, what provides higher transparency to the service architecture.

One of the main challenges in a hybrid setup is the image files management. The two use cases presented avoided to tackle that issue preparing the same images in the local infrastructure and the cloud providers manually.

The previous architecture has been used to implement two use cases: the deployment of a Sun Grid Engine computing cluster [5], and the implementation of a virtualized web cluster [6].

3.1.1. Sun Grid Engine Use Case

The work "Multicloud deployment of computing clusters for loosely coupled MTC applications" [5] applied the previously described hybrid cloud architecture to analyze the viability, from the point of view of scalability, performance and cost, of deploying large virtual cluster

infrastructures distributed over different cloud providers for solving loosely coupled Many-Task Computing (MTC) applications. In particular, to represent the execution profile of loosely-coupled applications, the chosen workload was the Embarrassingly Distributed (ED) benchmark from the Numerical Aerodynamic Simulation (NAS) Grid Benchmarks (NGB) suite [1]. The ED benchmark consists of multiple independent runs of a flow solver, each one with a different initialization constant for the flow field.

The first contribution of this work is the implementation of a real experimental testbed, consisting of resources from an in-house infrastructure and external resources from three different cloud sites: Amazon EC2 (Europe and USA zones) and ElasticHosts. On top of this multi-cloud infrastructure spanning four different sites, a real computing cluster testbed was deployed.

The experimental testbed starts from a virtual cluster deployed in the local data center, with a queuing system managed by Sun Grid Engine (SGE) software, and consisting of a cluster front-end (SGE master) and a fixed number of virtual worker nodes. This cluster can be scaled-out by deploying new virtual worker nodes, which can be deployed on different sites (either locally or in different remote clouds).

This scenario was used to analyze the performance of different cluster configurations, deploying the virtual worker nodes in different combinations of local and remote resources. For the cluster performance, the cluster throughput (i.e. completed jobs per second) was used as the metric. The resulting performance results prove that, for the MTC workload under consideration (loosely coupled parameter sweep applications), multi-cloud cluster implementations do not incur in performance slowdowns compared to single-site implementations, and showing that the cluster performance (i.e. throughput) scales linearly when the local cluster infrastructure is complemented with external cloud nodes. This fact proves that the multi-cloud implementation of a computing cluster is viable from the point of view of scalability, and does not introduce important overheads, which could cause significant performance degradation

In addition, a study to quantify the cost of these cluster configurations was conducted, measuring the cost of the infrastructure per time unit. The performance/cost ratio was also analyzed, showing that some cloud-based configurations exhibit better performance/cost ratio than local setup, so proving that the multi-cloud solution is also appealing from a cost perspective.

3.1.2. Nginx Use Case

The paper "Elastic management of cluster-based services in the cloud" [6] evaluates the performance and scalability of the hybrid cloud architecture described above for deploying a distributed web server architecture. For this purpose, two different cluster-based web server architectures were implemented on top of the OpenNebula-based hybrid cloud setup, using Amazon EC2 (US zone) as the remote provider.

The first web server cluster architecture considered in this work is a simple web server for serving static files, deployed on top of a hybrid virtual infrastructure. It consists of a server front-end that runs the Nginx reverse proxy software and distributes the user HTTP requests, using the round robin algorithm, among the different virtual back-end servers, which run the Nginx web server software. The virtual machines hosting these back-end servers can be deployed in the in-house physical resource pool or in Amazon EC2.

The VMs running on the local data center are deployed using the XEN hypervisor version 3.3.0, and have a 32-bit i386 architecture (equivalent to 1.0 GHz Xeon processor), 512 MB of memory, and Debian Etch OS. On the other hand, the remote VMs are based on an EC2 small standard instance (equivalent to 1.0-1.2 GHz Xeon processor), with 32-bit platform, 1.7 GB of memory, and Debian Etch OS. Although most modern high-performance web servers are typically based on 64-bit platforms, for testing purposes in this simple architecture the authors chose the most basic instances provided by Amazon (small 32-bit instances) and similar hardware configuration for local nodes.

In the network infrastructure implemented for the virtual web server cluster, every vir-

tual back-end node communicates with the front-end trough the private local area network. The local back-end nodes and the front-end are directly connected to this private network by means a virtual bridge configured in every physical host. On the other hand, the remote back-end nodes (deployed on Amazon EC2) are connected to the private network by means a virtual private network (VPN) tunnel, using the OpenVPN software. This tunnel is established between each remote node (OpenVPN client) and the cluster front-end (OpenVPN server). Although the OpenVPN tunnel can introduce some extra latencies in the communication between the front-end and the remote back-end nodes; it offers the advantage that since communications are encrypted, it is possible to implement a SSL wrapper in the front-end, so that it can decrypt HTTPS requests from client browsers and pass them as plain HTTP to the back-end servers (either local or remote), without compromising privacy.

This scenario is used to analyze the cluster elasticity and throughput when it is scaled out with a growing number of remote back-end nodes deployed on Amazon EC2. To measure it, the experiment is conducted with a fixed number of 300 client HTTP requests of static files of different sizes; using different cluster configurations with increasing number of nodes (from 4 to 24 nodes), with different combinations of local and remote nodes. Two metrics are obtained: the time needed to complete all the client requests; and the cluster throughput (number of requests served per second).

The graphics included in the paper show clearly that, in spite of the obvious extra communication overheads to the remote Amazon EC2 nodes, a sustained improvement in cluster throughput is obtained by adding an increasing number of remote nodes to the cluster, so proving the scalability of the proposed solution.

The second architecture considered in this work is a multi-tier server cluster adapted for the execution of the CloudStone benchmark. CloudStone, a project of the RAD Laboratory at UC Berkeley, is a toolkit consisting of an open-source Web 2.0 application, called Olio and a set of automation tools for generating load and measuring performance in different deployment environments [9]. The Olio project, launched by Sun and UC Berkeley, defines

a typical Web 2.0 social application (in particular, a social-event calendar web application) and provides three initial implementations: PHP, Java EE and Ruby-on-Rails (RoR). The toolkit also includes an open-source workload generator Faban, and defines ways to drive load against the application in order to measure performance.

In this work the authors choose the RoR version of CloudStone, motivated by the fact that the preferred architecture for RoR deployment follows a proxy server based architecture. Following the architecture proposed by Sobel et al [9], they implement a multi-tier web server cluster deployed on top of a hybrid virtual infrastructure which is composed by a front-end server (implemented by an Ngnix reverse proxy, with round robin load balancing), a variable number of virtual back-end nodes running the Rails application server processes (in particular, the Thin application server software), and a data base server (based on MySQL server). Again the back-end nodes are implemented as virtual machines, and can be deployed either on the in-house resource pool or remotely in Amazon EC2. In this case, the front-end and the database servers are also VMs deployed in the local data center.

This setup is tested using a single instance of the Faban-based workload driver. During a run, Faban records the response times of each request made by the load generator, from the time the request is issued by Faban until the time at which the last byte of the response is received. The request rate is measured between the initiations of successive operations. To prove different benchmark loads, all the experiments are run for different number of concurrent users. Similarly to the previous use case, the scalability of the setup is tested by deploying a growing number of application server nodes in the local and remote Amazon EC2 infrastructure, each one running two Thin processes (one Thin process per CPU core). For each configuration, the throughput (total number of completed operations per second) is analyzed, showing that the proposed architecture exhibits a good scalability when it is deployed in a single data center, but worsens when the number of remote Amazon EC2 nodes increases.

Although the architecture worked, the outcome of the tests leaded to the conclusion

that the hybrid setup does not result in a good scalability for this kind of highly interactive workloads with low time-out tolerance levels.

3.2. Hybrid Storage

In this section we introduce the different ways that OpenNebula has to interact with external Cloud Storage Providers in order to benefit from their services. Also we introduce a new proposal to add to OpenNebula the ability to download and register in the local architecture virtual machines from these remote cloud providers.

OpenNebula orchestrates storage, network, virtualization, monitoring, and security technologies using specific drivers for each task. In Figure 3.1 the internal architecture of the OpenNebula core is shown.

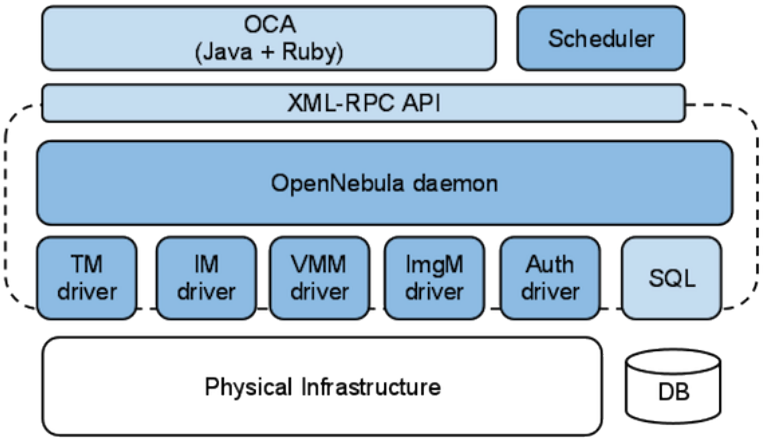

Figura 3.1: *OpenNebula architecture*

The main drivers of the OpenNebula core can be listed as follows:

- Transfer Manager (TM) drivers are used to transfer, clone and remove Virtual Machines Image files. They take care of the file transfer from the OpenNebula image repository to the physical hosts. There are specific drivers for different storage configurations: shared, non-shared, lvm storage, etc.

- Virtual Machine Manager (VMM) drivers translate the high-level OpenNebula virtual machine life-cycle management actions, like "deploy", "shutdown", etc. into specific hypervisor operations. For instance, the KVM driver will issue a "virsh create" command in the physical host. The EC2 or ElasticHosts drivers translate the actions into Amazon EC2 or ElasticHosts API calls.

- The Information Manager (IM) drivers gather information about the physical host and hypervisor status, so the OpenNebula scheduler knows the available resources and can deploy the virtual machines accordingly.

- The Image Manager (ImgM) drivers are quite similar to the Transfer Manager drivers. These drivers transfer the image files from their source location to the Image Repository when new images are registered. The Image Repository system allows OpenNebula administrators and users to set up images, which can be operative systems or data, to be used in Virtual Machines easily. These images can be used by several Virtual Machines simultaneously, and also shared with other users. This data can be then saved overwriting the original image, or as a new OpenNebula image.

  When a new OpenNebula Image is registered in the Image Repository, the current OpenNebula Image Manager driver performs a regular file copy if the source is a local filesystem, or if the source is a web or FTP server then it downloads the files and then copies them into the Image Repository.

In order to use images in OpenNebula from external providers we have two options:

1. Adapt the Transfer Manager drivers to interact with the external Cloud providers resources, each time an Image is needed in a host.

2. Import the image to the OpenNebula Image repository and deal with it like any other OpenNebula image.

The first option benefits from the OpenNebula modular architecture, leveraging the task of dealing with the Cloud provider to the transfer manager driver.

The last version of OpenNebula includes a transfer manager implementation to deal with http URLs in addition to the default behavior based on local paths. In this implementation the resource is specified by a unique URL. This URL that identifies the storage resource is included in the OpenNebula image template as the source parameter. Therefore, the image repository will not do any action on this resource. The Image will be copied to the repository only if a path parameter is specified in the template instead of a source.

If a Virtual Machine needs this resource the transfer manager will check the source value and will handle it in a different way, downloading the file from the specified URL instead of copying it from the local repository, as shown in the following code snippet

```
#!/bin/bash

SRC=$1
DST=$2

if [ -z "${ONE_LOCATION}" ]; then
    TMCOMMON=/usr/lib/one/mads/tm_common.sh
else
    TMCOMMON=$ONE_LOCATION/lib/mads/tm_common.sh
fi

. $TMCOMMON

get_vmdir

SRC_PATH=`arg_path $SRC`
DST_PATH=`arg_path $DST`

fix_paths

log_debug "$1 $2"
log_debug "DST: $DST_PATH"

DST_DIR=`dirname $DST_PATH`

log "Creating directory $DST_DIR"
exec_and_log "mkdir -p $DST_DIR"
exec_and_log "chmod a+w $DST_DIR"

case $SRC in
http://*)
```

```
    log "Downloading $SRC"
    exec_and_log "$WGET -O $DST_PATH $SRC" \
        "Error downloading $SRC"
    ;;

*)
    log "Cloning $SRC_PATH"
    exec_and_log "cp -r $SRC_PATH $DST_PATH" \
        "Error copying $SRC to $DST"
    ;;
esac

exec_and_log "chmod a+rw $DST_PATH"
```

Based on this concept we can easily provide OpenNebula the functionality to interact with external cloud storage providers. We just have to specify a new tag in the source parameter of the image template and implement a way to deal with these kind of resources in the transfer manager. For example if we want to interact with the Amazon s3 service we can define a source s3://s3.amazonaws.com/mybucket/myimage.manifest.xml and the transfer manager can use the EC2 tools (ec2-download-bundle, ec2-unbundle, etc.) to download the image.

Leveraging the interaction with the cloud provider to the transfer manager forces OpenNebula to download the image each time a new virtual machine is deployed. This is one of the biggest problems when dealing with external cloud providers, if the provider is not in the same domain or network, the time that a client spends downloading the resource can be unaffordable and even worst if this action has to be performed for each new virtual machine.

However, we are considering that the images have to be downloaded from the cloud provider but some storage solutions are prepared to export these storage resources. This is the case of the CDMI storage API which allows to mount an NFSv4 share or connect to an iSCSI or FibreChannel target the requested storage object. This option can also be supported by the OpenNebula core and its development can be based on the existing LVM transfer manager driver in which the device is offered to the virtual machine in the destiny path, as shown in the LVM implementation:

```
log "Creating LV $LV_NAME"
exec_and_log "$SSH $DST_HOST $SUDO $LVCREATE -L$SIZE -n $LV_NAME $VG_NAME"
```

```
exec_and_log "$SSH $DST_HOST ln -s /dev/$VG_NAME/$LV_NAME $DST_PATH"

log "Dumping Image"
exec_and_log "eval cat $SRC_PATH | $SSH $DST_HOST $SUDO $DD of=/dev/
    $VG_NAME/$LV_NAME bs=64k"
```

One of the advantages of leveraging this task to the transfer manager is that each new virtual machine will use the last version of the image stored in the provider. This situation can benefit the user providing for example an image with an updated version of a database for a new virtual machine, but also can cause that our new virtual machine fails to start due to an unstable or incomplete snapshot of the image.

The second option proposed above to deal with external cloud providers, importing the image to the OpenNebula repository, can mitigate some of the problems shown in the previous section. Using this solution the image will only be downloaded once and included in the repository, saving a lot of time if we want to use this resource in more than one virtual machine.

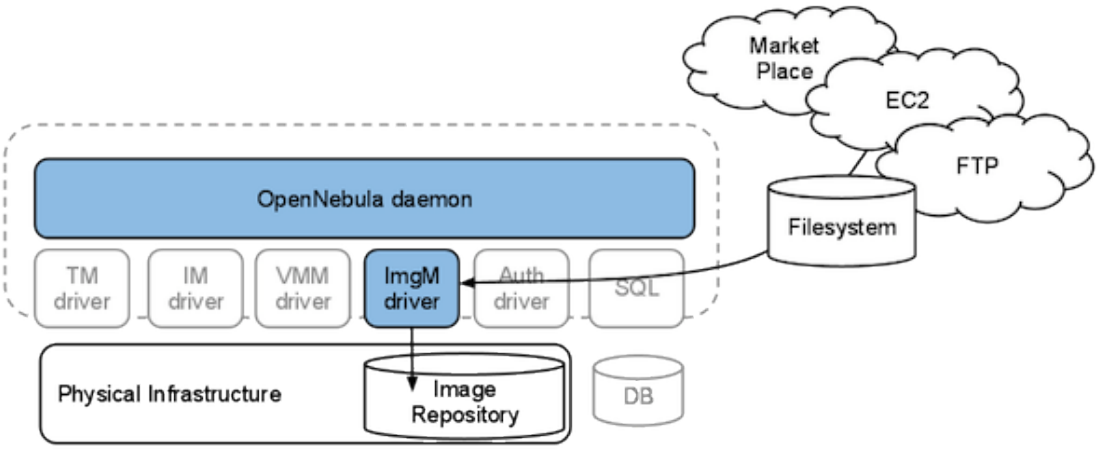

Figura 3.2: *Hybrid Cloud Storage*

As well as in the previous case OpenNebula allow us to easily implement this use case. In this solution the URL that identifies the resource must be defined as the path parameter instead of the source one, inside the image template. Therefore the image repository will download the file and will include it in the set of available local images. If a virtual machine

needs to use this image the transfer manager will copy it from the image repository instead of downloading it again.

As shown in the two previous examples OpenNebula can be easily extended in order to interact with any external cloud storage provider. We can even choose the best solution that fits our requirements between using the image repository or interacting with the external provider each time a virtual machine is created.

3.2.1. EC2 AMIs Use Case

In the previous section we have seen how to import images from external Cloud providers to OpenNebula, considering that these images were prepared to run in OpenNebula and no modification was needed. In this use case we will deal with images that are not prepared to run in OpenNebula such as the Amazon Machine Image (AMI) a special type of pre-configured operating system and virtual application software which is used to create a virtual machine within the Amazon Elastic Compute Cloud (EC2).

For this purpose a new component has been developed, based on the concept of catalogs. A catalog is a set of images offered by a Cloud provider or a partner. The high-level view of this new functionality is that OpenNebula will be able to manage a series of catalogs from where to list, import and adapt images to the local infrastructure. The administrators will be able to manage the available catalogs; and users will have the option to create a new virtual machine in OpenNebula from one of the remote images.

In order to run an external image in OpenNebula we identify two main steps: on the one hand, we have to download the image to the local repository, and on the other hand we have to adapt the image to benefit from the OpenNebula contextualitation.

Although the architecture is extensible with different plug-ins, the given example is centered in Amazon EC2, as it is the cloud provider with the most broadly used image catalog.

Amazon EC2 is used jointly with Amazon Simple Storage Service (Amazon S3) for in-

stances with root devices backed by local instance storage. In this example we are considering images with a root device backed by Amazon S3

The first step to use an AMI in OpenNebula would be to download the image from Amazon S3 so that we can modify it locally. In S3 all the objects are stored in buckets and the AMIs are bundled providing a manifest file that contains information of the bundling process and the AMI metadata.

As explained in the previous section we can modify OpenNebula in two different ways in order to retrieve images from external cloud providers. On the one hand we can modify the transfer manager and download the image each a new virtual machine is created. On the other hand we can extend the image repository and download the image to use it like any other local image. In this use case we have chosen the second options since we have to adapt the image before deploying it in any host, as we will see later. Also this option mitigates the amount of time spent in downloading images from S3, since this process is performed once per image instead of one per virtual machine.

In order to interact with the Amazon S3 API we can use the AMI tools provided by Amazon that will ease the process of downloading the image from S3. Therefore the image repository driver should be modified as follows. The S3 bucket that contains the target AMI is downloaded using the ec2-download-bundle command and the required credentials:

```
ec2-download-bundle -b mybucket -p myimage -a $EC2_ACCESS_KEY -s
    $EC2_SECRET_ACCESS_KEY -k $EC2_PRIVATE_KEY -d $DOWNLOAD_DIR
```

After that the image is unbundled in order to get a single file containing the whole image. For this, we will use the ec2-unbundle command that needs the EC2 private key that was used to bundle the image:

```
ec2-unbundle -m myimage.manifest.xml -k $EC2_PRIVATE_KEY -s $DOWNLOAD_DIR
    -d $DOWNLOAD_DIR
```

At this point a new image has been created in the OpenNebula repository and we would we able to start a new virtual machine based on this image if it was prepared to benefit from the OpenNebula contextualization that configures the network and runs user scripts.

However, as explained at the beginning of this section we are trying to import an EC2 AMI to OpenNebula instead of an OpenNebula image.

When you start a new virtual machine in Amazon EC2 specific user data information can be provided at start up in order to pass configuration information or even initialization scripts to the instance upon launch. This information is accessible from inside the instance through a metadata server, only accessible for the instance. Therefore this information can be retrieved using any utility such as curl performing GET requests on the server. In this server we can also find metadata information of the instance, as shown in the following requests:

```
GET http://169.254.169.254/latest/meta-data/
amiid
ami-launch-index
ami-manifest-path
block-device-mapping/
hostname
instance-action
instance-id
instance-type
kernel-id
local-hostname
local-ipv4
mac
network/
placement/
public-hostname
public-ipv4
public-keys/
reservation-id
security-groups

GET http://169.254.169.254/latest/meta-data/ami-id
ami-2bb65342

GET http://169.254.169.254/latest/user-data
#!/bin/bash
set -e -x
apt-get update && apt-get upgrade -y"
```

As explained in the previous sections, OpenNebula uses a CDROM device to provide specific instance information to the virtual machines; whereas the Amazon EC2 images expect a metadata server. Therefore the next step of the catalog component will be to

adapt this image to the OpenNebula contextualization.

The first step of this process would be to set up the OpenNebula network init script that will configure the interfaces of the instance. This script depends on the OS distribution and should be executed at boot time before starting any network service. You can find examples for Debian, Ubuntu, CentOS and openSUSE based systems in the OpenNebula installation

In this scripts you can find out that the OpenNebula IP address assigned to the virtual machine is derived from the MAC address using the MAC_PREFIX:IP rule. Whenever the virtual machine boots it will execute this script, which in turn would scan the available network interfaces, extract their MAC addresses, make the MAC to IP conversion and construct a /etc/network/interfaces that will ensure the correct IP assignment to the corresponding interface.

Having done so, a virtual machine based on this image could be started and it would be configured accordingly to the defined OpenNebula virtual network. But we are still missing the functionality of providing specific user data for each instances.

User specific information can be provided in OpenNebula instances using the CONTEXT section of the virtual machine template APPENDIX A. If OpenNebula detects this section in the template it will create a new ISO image containing the specified files and variables and it will be available from inside the instance in a CDROM device. We just have to mount the device as follows:

```
mount -t iso9660 /dev/sdc /mnt

if [ -f /mnt/context.sh ]; then
  . /mnt/context.sh
fi
```

It would be easy to add a new init script that used this information to configure the virtual machine at start up. However, this task is not necessary because, as we have explained at the beginning of this section, we are dealing with an EC2 AMI that is supposed to be prepared to interact with a metadata server to be customized.

Instead of creating new scripts to interact with the OpenNebula context ISO we will

serve this information in a local server to the instance, so that the instance will work as it was running in Amazon, combining both approaches without a substantial modification of the guest operative system. For this purpose we have chosen the SimpleHTTPServer provided by Python that allow us to serve the content of a directory in the desired URL and the CONTEXT section of the virtual machine will contain the information required to fill this directory. All this process will be included in a init script inside the instance that will be executed before the EC2 script, that tries to use the metadata server located in the url http::/169.254.169.254/ at boot time.

After that the image will be ready to be deployed in OpenNebula. The difference with other images is that a exact copy is still available in EC2, so the users can choose to deploy this virtual machine locally or in Amazon, without further configuration. The user will only have to specify the specific user data to configure the instance and the public key to access from the outside.

With this use case we have seen that we can move our images from the public to the private cloud or even combine them, in a hybrid cloud, without substantial efforts. Therefore we can customize our cloud on-demand, using the private or the public cloud when needed. We can run all our instances locally and if a host fails or we run out of capacity we can deploy the same instances in a public cloud enabling highly scalable hosting environments.

Capítulo 4

Conclusion

During this work we have studied the limitations and challenges that have to be faced in a Hybrid Cloud environment. Analyzing the different cloud interfaces that we can find; the options to interconnect virtual machines in different domains; the variety of image formats depending on the hypervisor; and the heterogeneous resources of each provider.

Also we have proposed solutions for all these challenges in three use cases: the deployment of a Sun Grid Engine computing cluster; the implementation of a virtualized web cluster; and the proposal of a new component to interact with storage cloud providers showing how we can import an EC2 AMI into OpenNebula.

This work has been used as a reference and a starting point for the development of a new component that will be added to OpenNebula in order to achieve a interoperable storage environment, being able to interconnect OpenNebula with market places and external cloud providers that will provide the image repository with resources. In addition to this development two chapters have been contributed as part of two different books that will be published in the following months.

The first chapter is titled "The OpenNebula Cloud Toolkit" [2] and is part of the book "Open Source Cloud Computing Systems: Practices and Paradigms" published by IGI Global scheduled for release in 2011. This chapter aims to describe the OpenNebula Cloud Toolkit, a framework that intends to provide a efficient and scalable solution for the management of Virtual Machines running on a pool of physical resources.

The second chapter is titled "On the use of the Hybrid Cloud Computing Paradigm" [3] and is part of the book "Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice" also published by IGI Global scheduled for release in 2012. This chapter analyzes the Hybrid Cloud computing model doing a review of the limitations, challenges and common techniques to address these challenges in this kind of deployments.

The conclusion that we can draw from this work is that the hybrid cloud is a model that is not yet fully developed, and there are still a lot of work to be done specially at the interoperability and standardization level. There are a lot of groups working on this matter such as the Open Grid Forum (OGF) group that are developing a standard API for managing resources, called the Open Cloud Computing Interface (OCCI), we can also find the Cloud Data Management Interface (CDMI) storage API or the Open Virtualization Format (OVF) standard to import virtual environments.

All these initiatives try to define a set of cloud standards for building interoperable cloud environments in the near future. This is the point where more time should be invested in the coming years, in order to avoid the variety of cloud interfaces, image formats, contextualization methods, etc that can be found in each cloud environment.

Bibliografía

[1] Michael Frumkin and Rob F. Van der Wijngaart. Nas grid benchmarks: A tool for grid space exploration. *Cluster Computing*, 5:247–255, 2002. 10.1023/A:1015669003494.

[2] Carlos Martín Sánchez, Daniel Molina Aranda, Jaime Melis, Javier Fontán, Constantino Vázquez, Rafael Moreno-Vozmediano, Rubén S. Montero, and Ignacio M. Llorente. The opennebula cloud toolkit. In *Open Source Cloud Computing Systems: Practices and Paradigms*. IGI.

[3] Carlos Martín Sánchez, Daniel Molina Aranda, Rafael Moreno-Vozmediano, Rubén S. Montero, and Ignacio M. Llorente. On the use of the hybrid cloud computing paradigm. In *Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice*. IGI.

[4] Miguel Matos, António Sousa, José Pereira, and Rui Oliveira. Clon: overlay network for clouds. pages 14–17, 2009.

[5] R. Moreno-Vozmediano, R.S. Montero, and I.M. Llorente. Multicloud deployment of computing clusters for loosely coupled mtc applications. *Parallel and Distributed Systems, IEEE Transactions on*, 22(6):924 –930, june 2011.

[6] Rafael Moreno-Vozmediano, Ruben S. Montero, and Ignacio M. Llorente. Elastic management of cluster-based services in the cloud. pages 19–24, 2009.

[7] Siani Pearson. Taking account of privacy when designing cloud computing services. pages 44–52, 2009.

[8] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K.Ñagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan.

The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):4:1 –4:11, july 2009.

[9] Will Sobel, Shanti Subramanyam, Akara Sucharitakul, Jimmy Nguyen, Hubert Wong, Arthur Klepchukov, Sheetal Patil, O Fox, and David Patterson. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. 2008.

[10] B. Sotomayor, R.S. Montero, I.M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *Internet Computing, IEEE*, 13(5):14 –22, sept.-oct. 2009.

[11] M. Tsugawa and J.A.B. Fortes. A virtual network (vine) architecture for grid computing. *Parallel and Distributed Processing Symposium, International*, 0:123, 2006.

Apéndice A

Step by Step Service Deployment in OpenNebula

The purpose of this section is to demonstrate how to quickly deploy a VM with OpenNebula in a few easy steps. We will assume that OpenNebula is properly configured and that at least one Host is available running KVM. Downloading the pre-configured VM

A contextualized VM image is available for download at dev.opennebula.org. The VM runs ttylinux.

```
$ cd
$ mkdir one-templates
$ cd one-templates
$ wget http://dev.opennebula.org/attachments/download/170/ttylinux.tar.gz
$ tar xvzf ttylinux.tar.gz
```

A.1. Preparing the Network

For this example we are going to use the simplest possible network configuration. We provide a file with the package which is a network template example, called small_network.net. Edit that file to change the LEASES entries to available IPs from the available physical network. If the Hypervisor is configured to use a different bridge, the BRIDGE entry should be also changed. Once the file is prepared we can create the network:

```
$ onevnet create small_network.net
```

A.2. Modifying the template

The VM template just downloaded, ttylinux.one, has a few parameters that should be adjusted:

```
source    = "/path/to/ttylinux.img",
files     = "/path/to/init.sh /path/to/id_dsa.pub",
username  = "opennebula"
ip_public = "x.x.x.x"
```

Please leave the CONTEXT section commented, though. Choose the ip_public field to be an IP which does not match any of the predefined LEASES of small_network.net (only for demonstration purposes). Finally copy the user's public key to the directory where the VM was downloaded. Be sure to rename id_dsa.pub to id_rsa.pub in the VM template if using an RSA key.

A.3. Running the VM

We are ready to deploy the VM. To do so simply do:

```
$ onevm create ttylinux.one
```

It will take a minute or so to copy the image to OpenNebula's var location and to boot up the system. In the mean time we can figure out what IP the VM will have so that we can ssh into it.

```
$ onevm show ttylinux|grep IP
  IP=192.168.1.6,
```

By now, the VM should be up and running:

```
$ onevm list
  ID     USER     NAME STAT CPU     MEM         HOSTNAME          TIME
   3 oneadmin myttyser runn   0   65536        localhost 00 00:06:49
```

Note: If the STAT attribute is not runn the logs can contain error messages about why it did not boot. These logs are located in /var/log/one/<id>/vm.log (vm specific log) and /var/log/one/oned.log.

We can ssh into the VM. The user is root and the password is password:

```
$ ssh root@192.168.1.6
Warning: Permanently added '192.168.1.6' (RSA) to the list of known hosts.
root@192.168.1.6's password:

Chop wood, carry water.
#
```

Thanks to the init scripts explained in the VM Contextualization section, the VM got automatically configured with an IP from the pool of LEASES defined by the Virtual Network associated to the VM template.

A.4. Running the VM again with CONTEXT

Shutdown the VM, edit the template to uncomment the CONTEXT section and deploy another instance. This time, we can ssh to the VM without entering a password, since the id_dsa.pub has been copied to the authorized_keys of both root and the username account defined in the template.

Apéndice B

Hybrid Cloud Deployment powered by OpenNebula

A Hybrid Cloud Deployment powered by OpenNebula is fully transparent to infrastructure users. Users continue using the same private and public Cloud interfaces, so the federation is not performed at service or application level but at infrastructure level by OpenNebula. It is the infrastructure administrator who takes decisions about the scale out of the infrastructure according to infrastructure or business policies.

There is no modification in the operation of OpenNebula to integrate Cloud services. A Cloud service is managed as any other OpenNebula host that may provide "infinite" capacity for the execution of VMs.

You should take into account the following technical considerations when using the EC2 cloud with OpenNebula:

- There is no direct access to the dom0, so it cannot be monitored (we don't know where the VM is running on the EC2 cloud).
- The usual OpenNebula functionality for snapshotting, restoring, or migration is not available with EC2.
- By default OpenNebula will always launch small instances, unless otherwise specified.

B.1. EC2 Configuration

You must have a working account for AWS and signup for EC2 and S3 services, and also download and unpack the EC2 API tools provided by Amazon. Do some manual test to verify everything works before start configuring OpenNebula for EC2 support.

B.2. Configuration

B.2.1. OpenNebula Configuration

Two lines must be added to the /etc/one/oned.conf file in order to use the driver.

```
IM_MAD = [
      name       = "im_ec2",
      executable = "one_im_ec2",
      arguments  = "im_ec2/im_ec2.conf",
      default    = "im_ec2/im_ec2.conf" ]

VM_MAD = [
      name       = "vmm_ec2",
      executable = "one_vmm_ec2",
      arguments  = "<ec2_configuration_options> vmm_ec2/vmm_ec2.conf",
      type       = "xml" ]
```

After configuring everything when you start ONE, you need to add the EC2 host to the host list to be able to submit virtual machines, like the following:

```
$ onehost create ec2 im_ec2 vmm_ec2 tm_dummy
```

B.2.2. Driver Configuration

Additionally you must configure the location of your EC2 certificates and EC2 API installation path, for this edit the file /etc/one/vmm_ec2/vmm_ec2rc and add:

```
EC2_HOME="<path_to_your_ec2_installation>"
EC2_PRIVATE_KEY="<path_to_your_private_key>"
EC2_CERT="<path_to_your_public_cert>"
```

Also you must configure the maximum capacity that you want OpenNebula to deploy on the EC2, for this edit the file /etc/one/im_ec2/im_ec2.conf, in this example we say that we want at much 4 small and 1 large instances launched into EC2:

```
# Max number of instances that can be launched into EC2

SMALL_INSTANCES=4
LARGE_INSTANCES=1
EXTRALARGE_INSTANCES=
```

B.3. EC2 Specific Template Attributes

Mandatory Attributes:

- AMI:the AMI name that will be launched
- KEYPAIR: This attribute indicates the rsa key-pair used to initiate the machines, the private keypair later will be used to execute commands like ssh -i id_keypair or scp -i id_keypair

Optional Attributes:

- ELASTICIP: This is the elastic IP address you want to assign to the instance launched.
- AUTHORIZED_PORTS: this parameter is passed to the command ec2-authorize default -p port, and must be in the form of a number (22) or a range (22-90),
- INSTANCETYPE: this attribute indicates the type of instance to be launched in EC2, by default all instances will be m1.small. Remember valid values for this are m1.small, m1.large, m1.xlarge, c1.medium, c1.xlarge.

B.4. Usage

You must create a template file containing the information of the AMIs you want to launch, its important to note that when deploying VMs on EC2 with OpenNebula, the template file should contain the attributes AMI and KEYPAIR used by the EC2 VMM Mad.

Additionally if you have an elastic IP address you want to use with your EC2 instances, you can specify it as an optional parameter.

```
CPU      = 0.5
MEMORY   = 128

#Xen or KVM template machine, this will be use when submitting this VM to
    local resources

OS       = [kernel="/vmlinuz",initrd= "/initrd.img",root="sda1" ]
DISK     = [source="/imges/apache.img",target="sda",readonly="no"]
NIC      = [bridge="eth0"]

#EC2 template machine, this will be use wen submitting this VM to EC2

EC2 = [ AMI="ami-00bafcb5",
        KEYPAIR="gsg-keypair",
        ELASTICIP="75.101.155.97",
        AUTHORIZED_PORTS="22",
        INSTANCETYPE=m1.small]

#Add this if you want to use only EC2 cloud
#REQUIREMENTS = 'NAME = "ec2"'
```

You can only submit and control the template using the OpenNebula interface:

```
$ onevm submit ec2template
```

Now you can monitor the state of the VM with

```
$ onevm list
  ID     USER      NAME STAT CPU     MEM         HOSTNAME          TIME
   0 oneadmin    one-0 runn   0       0               ec2 00 00:07:03
```

Also you can see information (like IP address) related to the amazon instance launched via the command

```
$ onevm show 0
VIRTUAL MACHINE 0 INFORMATION
ID                  : 0
NAME                : one-0
STATE               : ACTIVE
LCM_STATE           : RUNNING
START TIME          : 07/17 19:15:17
END TIME            : -
DEPLOY ID:          : i-53ad943a

VIRTUAL MACHINE TEMPLATE
EC2=[
  AMI=ami-acc723c5,
  AUTHORIZED_PORTS=22 ]
IP=ec2-174-129-94-206.compute-1.amazonaws.com
NAME=one-0
VMID=0
```