A Generic Catalog and Repository Service for Virtual Machine Images*

Jose V. Carrión, Germán Moltó, Carlos de Alfonso, Miguel Caballer, and Vicente Hernández

Instituto de Instrumentación para Imagen Molecular,
Universidad Politécnica de Valencia, 46022 Valencia, Spain,
{jocarbur,caralla,micafer1}@upv.es,{gmolto,vhernand}@dsic.upv.es

Abstract. The widespread usage of virtualization has caused a major impact in disparate areas such as scientific computing, industrial businesses and academic environments. This has led to a massive production of Virtual Machine Images (VMIs). The management of this broad spectrum of VMIs should consider the variety of operating systems, applications and hypervisors. This paper describes the developments towards a catalog and repository system for VMIs. Using the industry standard Open Virtualization Format (OVF) the system offers indexing and storage capabilities of VMIs as well as user-oriented matchmaking algorithms to leverage VMI sharing.

Key words: Cloud Computing, Virtualization, Catalog

1 Introduction

Virtualization has brought several benefits, such as reducing the number of physical machines by server consolidation and isolating the execution environments of applications. Using this as a base, Cloud technologies introduce the concept of computing as a utility, where it provides the illusion of an endless supply of computational resources available on demand [1]. The ability to access elastic computing platforms on a pay-per-use basis is of paramount importance for organizations, since these infrastructures can dynamically adapt to the workload requirements of such organizations.

There is currently no single and unified definition of Cloud technologies among the scientific community [2]. However, it is commonly agreed that Infrastructure as a Service (IaaS) Cloud deployments provide the greatest flexibility for generic application execution, when compared to other approaches such as Platform as a Service (PaaS) and Software as a Service (SaaS). An IaaS Cloud, such as the public infrastructure offered by Amazon EC2 [3], provides Virtual Machine (VM) management so that computational cycles can be consumed by

* The authors wish to thank the financial support received from the *Vicerrectorado de Investigación de la Universidad Politécnica de Valencia* for the project PAID-06-09-2810 and to the *Ministerio de Ciencia e Innovación* for the project TIN2010-17804.

applications. In addition, there exist software alternatives focused on deploying private, public and hybrid Clouds, such as Eucalyptus [4] and OpenNebula [5].

This has led to numerous Cloud deployments that provide VM management capabilities which, in turn, require creating application-specific VMs. These are typically known as Virtual Appliances (VA), which provide the application with the required environment for its successful execution. Creating a VA out of a VM requires a process called contextualization which, in many cases, can be automated.

Therefore, contextualizing, indexing and cataloguing the VMIs within an organization is important so that they can be reused for different projects. In addition, it could be interesting to share VMIs with external organizations that may find suitable to reuse existing VMIs to address similar problems, specially within the scientific community. This requires a three-fold strategy. Firstly, a catalog for indexing the VMIs with appropriate metadata to be used by matchmaking algorithms able to find the most appropriate VMIs to execute a given application. Secondly, a repository that efficiently stores and manages the VMIs, across different devices. Finally, a contextualization software able to automatically deploy an application on a VM ensuring the required execution environment.

The remainder of the paper is structured as follows. First, section 2 introduces the related work found in the literature. Next, section 3 describes the architecture and functionality of the catalog and repository system. Section 4 delves into the matchmaking algorithms employed to obtain a set of candidate VMs that satisfy the user requirements. Later, section 5 describes the approach employed to contextualize a VM for the successful execution of an application. Then, section 6 focuses on two applications in which we are using the catalog and repository system. Finally, section 7 summarizes the paper and points to future work.

2 Related Work

The idea of virtual machine cataloguing is not a new topic in the Cloud computing field. Nowadays, with the advent of Cloud technologies the storage and indexing of VMs are becoming crucial tasks in order to cope with the large number of VMs being produced. This also introduces the chance to share with the community generic VMs that can be reused for different purposes, thus leveraging scientific collaboration. For this reason, different VMI repository approaches have been currently developed. We describe the approaches that are close to our work.

VMware Marketplace [6], developed by VMware Inc. currently provides a set of 1410 virtual machines to be used with their own virtual management system. Their web-based interface introduces a categorization of the VMIs that depends on their target application (i.e., networking, collaboration and communication, applications infrastructure, etc.). In addition, it allows a simple keyword search in order to find the desired VMIs. The VMIs can be downloaded and uploaded by the registered users using the web interface provided. The Science Clouds Marketplace [7, 8], promoted by the University of Chicago, focuses on VMIs

that are useful for Grid computing and scientific applications. It is currently a very small repository available online (only eight VMIs listed), provides no search mechanisms and few metadata are stored. All the VMIs can be downloaded but it does not provide tools to upload or index your own VMI.

One of the largest public Cloud infrastructures is Amazon Elastic Compute Cloud (EC2) [3]. EC2 offers a repository of the Amazon Machine Images (AMIs) [9] able to be run in their IaaS Cloud. The images are hierarchically organized using different features (provider, operating system, region), but it does not provide search tools. However, it enables the users to create and upload new AMIs specifically designed for their applications. A related repository is the Cloud Market [10], which is a catalog of Amazon EC2 images. It scans Amazon EC2 and extracts VMI information (platform, kernel, etc.) considering common locations of image information for extra details. The users can claim their images and add more information about them. It provides some simple search tools to find the desired images specifying parameters such as platform, architecture and additional tags. However, it does not store the images, only information about them.

Most of the VMI repositories analyzed are designed to be used with a given hypervisor technology or are tied to a specific cloud system. They only provide very basic search tools, usually using a web interface, thus discouraging the use of APIs to perform automated advanced queries. These repositories are valid for an interactive scenario, in which a human examines the features of the VMs and then decides whether they are suitable or not for his/her application. However, creating mechanisms for selecting the most suitable VMI requires well structured metadata information.

Concerning the contextualization of VMs for the execution of scientific applications, there are different approaches in the literature. The Nimbus project [11] introduced the concept of VM contextualization. They created a service to gather and distribute runtime information of VMs mainly used to dynamically deploy virtual clusters. VMPlants [12] proposes a system based on DAGs to represent both the actual and the desired state of a VM as the result of the installation of software packages and user actions. These graphs can be enacted on the VM to perform the specified actions. There also exist different tools to help performing actions unattended such as Chef [13] and Puppet [14], which can certainly be incorporated in a tool like the one proposed. In our case, we wanted to focus on the specific workflow required to deploy scientific applications.

This paper describes a Virtual Machine catalog and repository system, which is a service that indexes VMs with metadata that can be used for automated purposes. That information includes not only data about virtual hardware configuration (memory, processor, disk, etc.), but also operating system related (type of operating system, version, release, etc.) and applications installed.

3 The Virtual Machine Catalog and Repository Manager

The main goal of the Virtual Machine Catalog and Repository system (VMCR) is to enable users to upload, store and catalog their VMs so that they can later be searched and retrieved, possibly by other users in order to enhance sharing and collaboration. For that, we have relied on industry standards such as the Open Virtualization Format (OVF) [15], to describe the VMs, and Web Services to develop the core of the VMCR service itself.

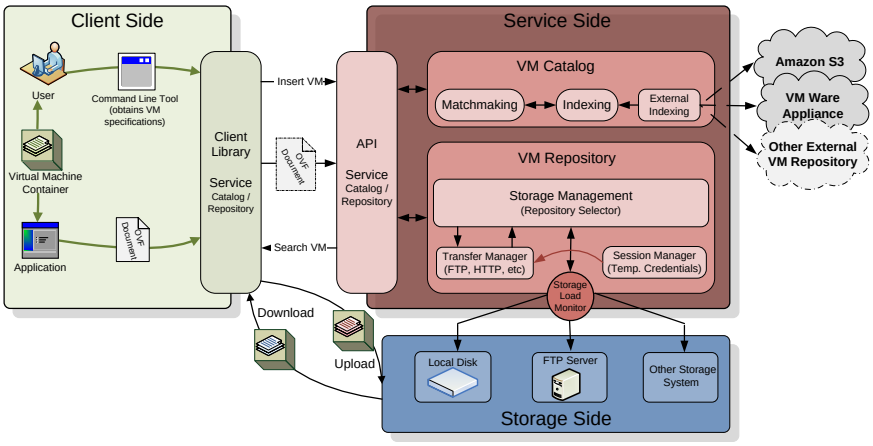

Fig. 1. Architecture of the Virtual Machine Catalog and Repository Manager

The architecture of the system is depicted in Figure 1. Its functionality is divided in four key areas (Client, Catalog, Repository and Storage). The *Storage* module is in charge of abstracting the details of different storage media, such as local disks and FTP servers, where the VMs will be stored. The *Repository* module provides support for different transfer protocols to be used by the client, such as FTP and HTTP. It also offers session management capabilities to issue temporary credentials employed to authorize the VM uploads. The *Catalog* module is responsible for indexing the VMs stored in the repository. Besides, it offers matchmaking algorithms that are employed to obtain the most appropriate VMs that satisfy the requirements imposed by the user.

Both the catalog and the repository expose a well defined API in order to use its functionality from a programming language (currently Java bindings are offered). However, the *Client* module also includes end user command line applications in order to ease its usage by the user.

The next sections describe in detail the functionality and implementation of the catalog and repository modules.

3.1 Cataloguing Virtual Machines

The main functionality of the VMCR system is cataloging virtual machines. As such, the users can register their own VMIs in the catalog as well as its related metadata. This includes information about the virtual hardware configuration (memory, processor, disk, etc.), the operating system (type of operating system, version, release, etc.) and the applications installed. Including VM metadata enables to develop matchmaking algorithms, described in section 4, that can search in the catalog for the most appropriate VMIs according to user requirements.

Developing a generic catalog system for VMIs requires the system to be hypervisor-agnostic, so that it can be used as part of virtually any kind of Cloud environment. There are currently different hypervisor technologies being used by the community, such as KVM (Kernel-based Virtual Machine), Xen and VMware, each one using a different format to describe the VMs. This is why we are relying on industry standards such as the Open Virtualization Format (OVF) [15], developed by The Distributed Management Task Force.

The OVF format is a platform independent, efficient, extensible, and open packaging and distribution format for virtual machines An OVF package consists of several files, placed in one directory. The main files that compose the OVF package are the disk images, and the OVF descriptor. The descriptor is an XML file that describes all metadata for the virtual machines (including virtual hardware), as well as the structure of the OVF package itself.

The OVF descriptor file is able to define an extensive set of metadata about all the aspects of the VM: VM type, hardware features, operating system, applications installed, etc. Being based on the XML language, the standard can easily be extended in order to add new sections or VM capabilities. In particular, extra fields have been added to some of the information nodes to include detailed information. For example, the *OperatingSystemSection* node only provides two general fields (Info and Description), and it has been complemented with three new fields (Type, Flavour and Version) enabling to better describe the OS used in the VM. In addition, the *VirtualHardwareSection* now requires a new field (Quantity) that indicates the disk and memory size, and the number of CPUs for a virtual machine.

The catalog system provides indexing capabilities for the VMIs. Being a generic solution, it can be employed to catalog VMIs stored in external repositories. This way, it is possible to take advantage of the storage facilities of external providers and rely only on the functionality of the catalog.

The catalog service interface is exported as a Web Service, providing a set of well-defined APIs to access its functionality. It enables the user to exchange the VMI information using OVF-compliant XML documents. Relying on plain Web services allows to create a lightweight, easy-to-deploy system that is able to seamlessly run on an application server like the Apache Tomcat.

3.2 Repositories of Virtual Machines

The repository service supports the efficient storage and retrieval of the files related to the VMIs indexed by the catalog service. The internal storage structure of the repository is defined by means of internal directories called *containers* that are able to store an unlimited amount of files.

When an upload is requested to the repository the storage manager creates a new container and assigns it a unique identifier. This number is used by the client-side library to upload all the VM disk image files into the container. Similarly, when a download operation is requested this identifier allows to retrieve the list of files in the container in order to download them all. In addition, the catalog service links each virtual machine with the unique identifier of the container obtained from the repository.

The use of identifiers to manage the repository allows to decouple the catalog and the repository, since the repository is not aware of the virtual machine that is related to a given container. Therefore, this approach allows the user to store the files related to virtual machines regardless of the hypervisor employed. This guarantees extensibility since the repository can be used to store VMI created with other virtualization softwares.

Another important feature that should be pointed out is its ability to store a high volume of data, since the size of a VMI is in the order of a few gigabytes. Therefore, the service has been designed following a plugin-based approach in order to simultaneously use different devices to store the virtual machines. This allows the service to seamlessly use different types of storage devices, such as local disks, remote FTP servers and even Storage Area Networks (SAN). This only requires the development of the plugin for the specific storage device. This approach allows introducing mechanisms to choose the most appropriate storage device considering features such as data transfer speed or capacity requirements. Load balancing the storage leverages the scalability of the developed system.

Finally, the repository service enables the implementation of synchronization mechanisms among the files in different repositories. As replication and distribution are key elements for fault tolerance, this approach paves the way to have multiple repository services in order to better cope with failures and prevent data loss from the information stored in the repository.

According to Figure 1, the most important components included in the repository service are:

- API: Defines the main operations, which allow the user to upload and download the files of a VMI.
- Storage Manager. It is the central component of the service that interacts with the rest of the modules after an upload or download operation. It chooses the most appropriate storage device and the protocol to be employed during the data transfer considering the information offered by the Storage Load Monitor.
- Storage Load Monitor: It monitors the storage capacity of each device and reports this information to the Storage Manager.

– Transfer Manager: It chooses the transfer protocol to be used for sending and receiving the data files of the VMI. Depending on the protocol employed, it is necessary to generate certain temporary credentials, through the Session Manager, in order to authorize users to access the repository.
– Session Manager: Manages the required credentials so that the clients are able to upload and download the VMI.

The repository service currently uses an FTP service that is dynamically started upon a user's request. The credentials (users/passwords) of this service are also dynamically created and offered to the client-side library in order to lease authorizations to users. The FTP server typically configures the sessions so that clients can only access their VMI files. However, it is possible to provide special authorizations so that different users can access the files in a given container of the repository. The Session Manager might also use an authorization mechanism based on X.509 certificates that uniquely identify each user. This way, this component should have the credentials of the Certification Authority (CA) that issues the user certificates, together with an Access Control List that prevents unauthorized users to access the repository.

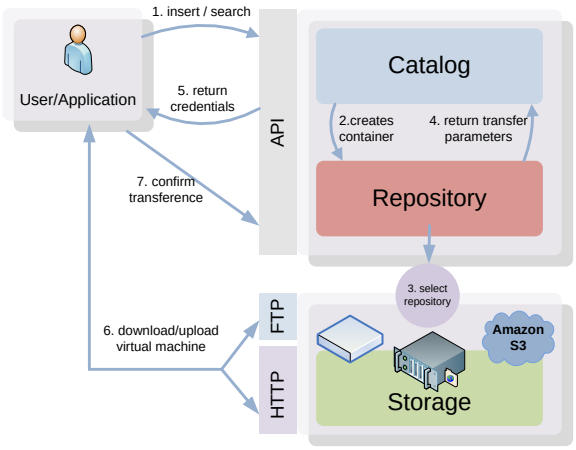

Fig. 2. Interaction diagram between the repository and the catalog services.

The repository service interacts with the catalog service as depicted in Figure 2. In case the user wants to upload a new virtual machine, the interaction proceeds as follows. First of all the client application invokes an operation in the catalog service to send the OVF document that specifies the virtual machine description (step 1). Then, the catalog service asks the repository component for a free container to store the virtual machine (step 2).

Later, the repository chooses the most appropriate storage device considering their storage load. The transfer protocol is typically chosen by the user from those supported by the service. Next, the repository service generates temporary

credentials that are be employed by the client to access the container in which the VMI files can be uploaded (step 3). Then, the catalog service receives the container identifier that relates the virtual machine with its repository (step 4). In step 5, the service provides the user with the credentials required to access the storage device. These credentials might vary depending on the storage device.

Finally, the client uses the credentials in order to directly access the storage device for file upload. After finishing the data transfer operation, the client informs the service that the data transfer has finished. Finally, to ensure that no errors were introduced during the data transfers, the client sends to the service the MD5 checksums of the uploaded files. This way, data can be checked for inconsistencies.

In the case of a download operation, the repository obtains the identifier of the container that hosts the virtual machine to be retrieved, and the client-side library is in charge of downloading the files of the VMI.

4 Matchmaking: Satisfying the User Requirements

A key issue when dealing with the VMCR system is to provide the user with matchmaking facilities. Otherwise the VMCR would be just another catalogue of VMIs. The matchmaking process aims at providing the users with the VMIs that best fit their application requirements.

When considering the VMI features that the user might require, they can be classified in two major groups (according to their importance for the user): hard and soft requirements. On the one hand, we have identified requirements that a VMI must fulfill to be appropriate for an application. Typical examples of these *hard* constraints are the operating system installed on the VMI, the amount of memory or the size of the hard disk. Furthermore, these requirements have to be extensible to express the applications or libraries that a VMI should have installed.

On the other hand, there are optional features that the user application might like the VMI to have them, but they are not a must. An example of a soft requirement would be the amount of memory that exceeds a given threshold (an application might require 512 Mbytes of RAM but its performance could probably be better with memory over 1 Gbyte). Another example is a certain application or library installed in the VMI, that it would be ideal to have it installed, but could certainly be installed afterwards. In this case, the VMI should definitely be a candidate choice for running the application. In the end, the user is responsible for deciding which attributes are considered hard of soft requirements.

According to these definitions it is clear that the user must not be provided with a VMI that does not satisfy every hard constraint expressed, but there probably exists more than one VMI that satisfies the soft requirements.

In order to establish some criteria for the user to understand how a VMI satisfies the requirements, we introduce the *Value of Interest* (VOI) concept for a requirement. It represents a rank value, expressed in a user-defined scale, for a

specific soft constraint stated by the user. Obviously the VOI has no sense for a hard requirement, as the VMI must be discarded if any of the hard requirement of an application is not met. In summary, the VOI stands for the niceness of a feature considered for the desired VMI, and this is certainly related to the application.

Searching for the candidate VMIs for an application requires the user constraints to be expressed in a custom language which is inspired by the Condor classads language [16]. In addition, it allows the user to specify whether each requirement is hard or not, and its VOI in the case of a soft requirement. An example set of simple assertions for a VMI are stated below:

```
os.type='linux' && os.flavor='ubuntu', hard
system.memory >= 512, hard
system.memory > 1024, soft, 100
```

Fig. 3. Sample language employed by the user to indicate the requirements that should be met by a Virtual Machine

The assertions in Figure 3 indicate that the user wants a VMI that has installed Ubuntu GNU/Linux and at least 512 Mbytes of RAM, although it would be better if it has $>$ 1 Gbyte of RAM. Using the provided VOIs, a mathematical function can be applied to rank a specific VMI. The result of that function will be called the *Suitability Value* (SV) for a VMI considering a set of requirements. Once the SV is calculated for every VMI that accomplishes the hard and some (if any) soft requirements, the results can be sorted out so that the user can determine which one of the VMIs best fits for the application. The function to calculate the SV of a VMI may vary according to the specific deployment of the VMCR. It can be as simple as the sum of every satisfied VOI or more complex, for example considering different weights for the VOIs, according to different sets of constraints (processor, memory, applications, etc.). This would give more importance to a specific set of requirements in contrast to others.

For the sake of completeness, it is important to point out that depending on the hypervisor employed, the memory of the VM would not be required to be a hard requirement, since this value can certainly be modified prior to booting the VMI. This is the case of, for example, the KVM hypervisor.

5 Contextualization and Virtual Machine Management

The execution of scientific applications in a Cloud computing infrastructure requires the creation of Virtual Appliances (VA). A VA is a VM that includes the application and the required environment for its execution (numerical libraries, databases, runtime environments, etc.), thus obtaining a self-contained application.

Configuring a VM to obtain a VA is typically called contextualization. Applications with a large number of dependencies on external software components (such as libraries, database management systems, etc.), cannot perform the contextualization at deployment time. Additionally, in most cases, performing automatic contextualization requires considerable complexity from a technical point of view. For these cases, a possible approach is performing the installation of the most complex software components by the user, in order to produce a pool of partially contextualized VMs which would be stored in the repository and indexed in the catalog service described in this paper. Later, these VMs would then be completely contextualized at boot time in order to create the appropriate environment required for the execution of the scientific application.

5.1 Contextualizing Scientific Virtual Appliances

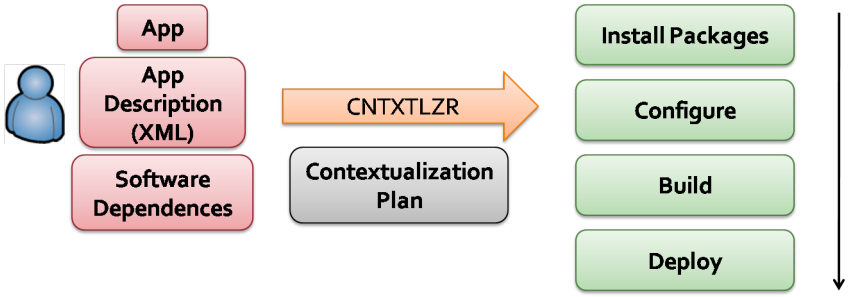

Fig. 4. Contextualization approach of Virtual Machines to Deploy Applications

We are working on a tool that enables to deploy scientific applications with minimal user intervention [17]. The main idea is to automatically perform the typical steps required when deploying a scientific application (packaging, configuration, compilation, execution). This way, instead of manually configuring the VM we favor application inoculation into the VM with minimal user intervention.

Figure 4 describes the approach employed by this software. With this tool, the users just expose in a declarative manner the deployment requirements of their applications, using an XML high-level based declarative language. Then, the tool automatically performs the steps involved in the deployment of a scientific application:

1. Install the required packages. Resolve dependencies with other software components and install those dependencies first.
2. Configuration. Enables the user to express the configuration process of the software package. This is achieved by common actions such as, copy files, change properties in configuration files, declare environment variables, etc.

3. Build. Compile the software package using the appropriate build system (Configure + Make, Apache Ant, SCons, etc.)
4. Execution. If required, start the application. This step might require invoking a shell script, performing a parallel execution, delegating the execution into a Grid system, etc.

Notice that, in general, once the VM is properly contextualized for a given application, it can be stored for later use. Therefore, depending on the application, the contextualization process might only be performed once and later reuse the VM for subsequent executions. Nevertheless, as different Cloud infrastructures might use different hypervisors, reusing VMs across different platforms is not currently a straightforward process.

5.2 Contextualization-aware matchmaking

We are also considering advanced matchmaking algorithms related to the contextualization of the virtual machines. These algorithms assume the existence of a service able to contextualize a VMI. If we assume that after the contextualization the resulting VMI is going to satisfy more requirements than the original one, we should consider that a constraint will be satisfied if we have a plugin that will achieve it. As a result, in the presence of a contextualization service, some VOIs should be taken into account.

The key issue here is that contextualization implies a post-process that can be both time consuming and have an associated cost. Therefore, the time involved in this process should include, on the one hand, the Expected Time to Contextualize (ETC), but also other hidden costs such as obtaining the appropriate licenses for proprietary software. In our developed model, we are only considering the ETC, as the other costs imply alternative mechanisms such as accounting, paying, etc. that should be implemented according to the specific business model.

Currently we have a contextualization mechanism that is plugin-driven [17]. The main objective is to have a plugin to get a specific feature (from reconfiguring the amount of disk to installing specific software). For each plugin i, its creator should be able to estimate the cost (typically the average installation time) for that plugin (Cp_i). Estimating the cost of the plugin could certainly be complemented by considering other issues such as the time needed to gather the software required to install. At this point we have a total cost to contextualize one specific VMI to meet some requirements. That cost (C) is calculated as the sum of the Cp_i for each plugin that should be used. The pseudocode of the contextualization-aware matchmaking algorithm is included in Figure 5.

The outcome of the algorithm is a list of VMIs that meet the requirements, or would meet them at a given cost. Then the user must evaluate whether to use the contextualization process or simply getting a VMI as is.

```
images={}
for each VMI as current
        C=0
        SV=0
        for each requirement as req
                if (!current satisfies req)
                        if (!we have a plugin to satisfy req)
                                if (req is hard)
                                        discard VMI and continue
                                endif
                        else
                                push(Cost(plugin), C)
                                if (req is soft)
                                        push(VOI(req), SV)
                        endif
                else
                        if (req is soft)
                                push(VOI(req), SV)
                end if
        end for
                push(current, SV, C, images)
end for
result = ordered images according to VOI
```

Fig. 5. Contextualization-aware matchmaking pseudocode

6 Case Study: Cataloguing Scientific Virtual Appliances

One of the main goals of the catalog is its application for the categorization of scientific virtual appliances. This section summarizes sample areas in which this system is being used.

6.1 Grid Service-based Virtual Appliances

Grid computing technologies [18] based on the Globus Toolkit 4 (GT4) [19] allow the development of Grid services which are loosely coupled interoperable services, such as Web services, but with the ability to maintain a persistent state. In order to introduce fault-tolerance for Grid services, Moltó et al. developed a pluggable library that automatically introduced replication capabilities for GT4-based Grid services [20]. This approach was later extended in order to allow the automatic deployment of virtual machines [21] which can be properly contextualized at boot time.

In order to deploy Grid services based on GT4, we are using a virtual machine based on Ubuntu JeOs 8.04 with a pre-installation of GT 4.0.8. The installation of all the required components (Java VM, C compiler, etc.) results in a VMI size of 1.2 Gbytes.

The usage of the catalog and the repository service allows to store and categorize common purpose virtual machines. The GT4-based VMI is stored in the catalog with the required metadata related to the hardware configuration (1 CPU, 512 MBytes of RAM, KVM hypervisor) and the applications installed (GT4, Java). This VM can be properly contextualized at boot time in order to deploy the specific Grid service. Therefore, the VMI can be reused for different Grid applications.

6.2 Virtualizing a Grid Infrastructure based on the gLite Middleware

With the evolution of the Enabling Grids for E-sciencE (EGEE) project [22] and now with the transition to European Grid Initiative (EGI) [23], the deployment of nodes based on the gLite middleware is a requirement for resource providers.

There are different components in a gLite deployment, each one requiring a different configuration. Moreover, the active development and enhancement of the middleware results in complex system administration tasks in order to maintain an updated gLite deployment. The most important components involved in a gLite site are the Computing Element (CE), the Storage Element (SE), and the Working Nodes (WN). Each gLite node (version 3.1) requires Scientific Linux 4 installed, with the specific configuration shown in Table 1 for each kind of node. Due to this OS restriction a common strategy consists of deploying the gLite nodes based on VMs.

Node Type	Memory (MB)	Disk (GB)	Software Packages
SE	1024	250	glite-SE_dpm_disk glite-SE_dpm_mysql
CE	512	8	lcg-CE or glite-CREAM
WN	512	4	glite-WN

Table 1. Requirements of different gLite node types.

The installation and configuration process of the different gLite nodes is quite similar. First, the gLite middleware (lcg-CE, glite-SE or glite-WN software packages) has to be installed and then each component has to be configured by means of the gLite *yaim* tool. The usage of the VMRC service, together with the contextualization tool aims at simplify the management of this infrastructure for a give site. The procedure follows this steps:

1. Contact the repository to find the most suitable VMI. A hard requirement is that the VMI should have the Scientific Linux 4 OS installed. However, to have installed the corresponding gLite software (SE, CE or WN) is considered as a soft requirement.
2. If the VMI has the gLite middleware installed, go to the next step; in other case install the corresponding gLite software package. Once installed the

gLite software package, the VM can be registered in the catalog to enable its reuse in the future. This last step is optional.
3. Configure the node using the gLite yaim configuration tool. This step involves the personalization of the node to the match the site's requirements as well as the organization policies.

By cataloguing the VMIs employed in a repository, it is easy to automate the deployment of new nodes in these kind of infrastructures.

7 Conclusions and Future Work

This paper has presented a catalog service that introduces a mechanism to properly classify and identify Virtual Machine Images. The catalog includes matchmaking algorithms in order to obtain a set of candidate virtual machines adequate for the deployment of a given application. In addition, extensible mechanisms are provided for the definition of additional matchmaking policies. The catalog interacts with a repository service that manages the efficient storage of virtual machines through the aggregated use of different storage devices. The catalog is currently being employed for different scientific applications.

Future works include connecting the virtual catalog to external repositories of VMIs such as those provided by Amazon EC2. Moreover, with the advent of virtual to virtual technologies, which allows migrating VMs among different hypervisor technologies, it would be interesting to create gateways to import VMs from external repositories into the local one. This way, it would be possible to avoid the hypervisor lock-in and take advantage of virtual machines originally created for other virtualization technologies.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley (2009)
2. Youseff, L., Butrico, M., Da Silva, D.: Toward a Unified Ontology of Cloud Computing. In: Grid Computing Environments Workshop (GCE '08). (2008) 1–10
3. Amazon: Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2` (2010)
4. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The Eucalyptus Open-source Cloud-computing system. In: Proceedings of 9th IEEE International Symposium on Cluster Computing and the Grid. (2009)
5. Sotomayor, B., Montero, R.S., Llorente, I.M., Foster, I.: Capacity Leasing in Cloud Systems using the Opennebula Engine. In: Workshop on Cloud Computing and its Applications 2008 (CCA08). (2008)
6. VMWare: Virtual Appliance Marketplace. `http://www.vmware.com/appliances` (2010)

7. Keahey, K., Figueiredo, R., Fortes, J., Freeman, T., Tsugawa, M.: Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. In: Cloud Computing and its Applications. (2008)
8. Clouds, S.: Science Clouds Marketplace. http://sciencecloudsorg/marketplace (2010)
9. Amazon: Amazon Machine Image (AMI). http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=171 (2010)
10. The Cloud Market. http://thecloudmarket.com/ (2010)
11. Keahey, K., Freeman, T.: Contextualization: Providing One-Click Virtual Clusters. In: Fourth IEEE International Conference on eScience. (2008) 301–308
12. Krsul, I., Ganguly, A., Fortes, J., Figueiredo, R.: VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. In: Proceedings of the ACM/IEEE SC2004 Conference, IEEE (2004) 7–7
13. OpsCode: Chef. http://www.opscode.com/chef (2010)
14. Labs, P.: Puppet. http://www.puppetlabs.com (2010)
15. DMTF: The Open Virtualization Format Specification, version 1.1.0. http://www.dmtf.org/standards/published_documents/DSP0243_1.1.0.pdf (2010)
16. Raman, R., Livny, M., Solomon, M.: Matchmaking: Distributed Resource Management for High Throughput Computing. In: In Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing. (1998) 28–31
17. Moltó, G., Hernández, V.: Management and Contextualization of Scientific Virtual Appliances. In: Cloud Futures 2010: Advancing Research with Cloud Computing. (2010)
18. Foster, I., Kesselman, C.: The GRID 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann (2004)
19. Foster, I.: Globus Toolkit version 4: Software for Service-oriented Systems. IFIP International Conference on Network and Parallel Computing, Lecture Notes in Computer Science **3779** (2005) 2–13
20. Moltó, G., Hernández, V., Alonso, J.M.: Automatic Replication of WSRF-based Grid Services via Operation Providers. Future Generation Computer Systems (International Journal of Grid Computing) **25**(8) (2009) 876–883
21. Moltó, G., Hernández, V.: On Demand Replication of WSRF-based Grid Services via Cloud Computing. In: 9th International Meeting on High Performance Computing for Computational Science. (2010) To appear.
22. Enabling Grids for e-SciencE: EGEE project (2010) http://www.eu-egee.org/.
23. European Grid Initiative: EGI project (2010) http://web.eu-egi.eu/.