On Demand Replication of WSRF-based Grid Services via Cloud Computing *

G. Moltó[1] and V. Hernández[1]

Instituto de Instrumentación para Imagen Molecular.
Universidad Politécnica de Valencia. Camino de Vera s/n 46022 Valencia, Spain
{gmolto,vhernand}@dsic.upv.es
Tel. +34963877356, Fax +34963877359

Abstract. With the introduction of the Globus Toolkit 4 (GT4), the service-oriented paradigm fully entered the Grid Computing arena. Since then, Grid services have turned into a suitable technology for building loosely coupled components for Grid deployments. Grid services require developers to consider fault tolerance and high availability as critical design factors to ensure successful long-term stability. This paper describes the integration of a library that allows to automatically replicate GT4-based Grid Services with a Cloud Computing back-end. Using an Infrastructure as a Service approach, the Cloud can provision GT4-based preconfigured virtual machines which are properly contextualized at boot time to deploy new Grid service replicas. This scheme allows replicated Grid Services to gain on demand scalability and increased fault tolerance.

Topics Parallel and Distributed Computing.

1 Introduction

The Globus Toolkit 4 (GT4) [1] allows developers to produce service-oriented components for the Grid via its implementation of the WSRF (Web Services Resource Framework) [2]. These loosely coupled Grid Services, as any distributed system, require fault-tolerant techniques to provide users with high availability. This way, replication stands out as a suitable approach to guarantee service provision via multiple coherent instances of the same Grid Service.

In a previous work [3] we described a library that manages the automatic replication of WSRF-based Grid Services. We also described the integration of the replication library into a service-oriented job metascheduler in order to enhance fault-tolerance and to guarantee service availability. This library allows

* 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 project *ngGrid: New Generation Components for the Efficient Exploitation of eScience Infrastructures (TIN2006-12890)*. This work has been partially supported by the Structural Funds of the European Regional Development (ERDF).

the automatic creation of groups of replicated Grid services, that is, the same Grid service replicated in different GT4 containers. It provides the services with operations to achieve coherent state replication so that all the replicas share the same state. In addition, the library exposes operations for group management, so that new replicas can join and leave the group, a feature specially important to replace the failed ones. However, this library required the user intervention for a new Grid service replica to join the replication group, since this involved the deployment of a new GT4 container as well as the replicated Grid service.

With the advent of Cloud Computing technologies, virtualization has become an ideal technology to deploy virtual machines that can be properly contextualized at boot time in order to fit almost any possible scenario. The usage of a Cloud infrastructure to automatically deploy virtual appliances hosting a GT4 container opens up a new scenario for on demand scalable replication of Grid services. However, this approach introduces new problems. First of all, virtual machines have to be properly configured so that they act as a new GT4-based replica. This involves a process commonly denoted as contextualization, that is, obtaining a fully configured virtual appliance from a base virtual machine. Secondly, the dynamic provision of virtual appliances benefits from relying on virtual machine managers, which manage most of the life cycle of virtual machines (deployment, monitoring, migration, termination, etc.). Finally, with the introduction of a virtualized layer, it is important to measure the penalties introduced by this approach, specially when the provision of new replicas aims at replacing the failed ones to maintain service availability. This paper reflects experiences on these three major topics.

The idea of service provision from Cloud infrastructures is not a new one. In [4], the authors describe the dynamic provision of computing resources from a Nimbus [5] cloud deployment. This way, they are able to increase a Grid infrastructure by including new worker nodes on demand. In [6], the authors address the issue of automatic infrastructure provision in order to create dynamic virtual clusters configured with a required software execution environment. They apply their approach into a metascheduler able to dynamically deploy Grid components. An architecture for resource virtualization based on Service Level Agreements is presented in [7], where users specify their resource requirements and brokers enact on demand service provision by means of virtual appliances.

As opposed to previous works, our approach does not focus on enlarging a computational infrastructure but rather on WSRF-based service provision. Notice that it is possible to have several spare machines with GT4 containers ready to be employed in case new replicas are requested. However, this approach requires dedicate computing resources which results in an overall underutilization of the machines considered in the infrastructure. Instead, using Cloud computing allows provisioning these resources on demand, whenever they are requested. In this paper we describe the integration of the Grid services replication library with a Cloud computing-based infrastructure. This way, new Grid service replicas can be deployed on demand to guarantee service provision.

The remainder of the paper is structured as follows. First, Section 2 describes the replication library. Next, section 3 covers the Cloud computing technologies employed. Section 4 addresses the process of contextualization, in order to create Virtual Appliances from base virtual machines. Then, section 5 describes the integration of the replication library with the Cloud back-end. Later, section 6 discusses the main overheads introduced by this approach. Finally, section 7 summarizes the paper and points to future work.

2 The WSRF-based Grid Service Replication Library

GT4 provides the implementation of a set of Grid services that are, basically, stateful Web services. WSRF specifies the way these Web services hold a state. This is achieved by coupling a data container, which stores the stateful data, to a Web service, thus obtaining a WS-Resource.

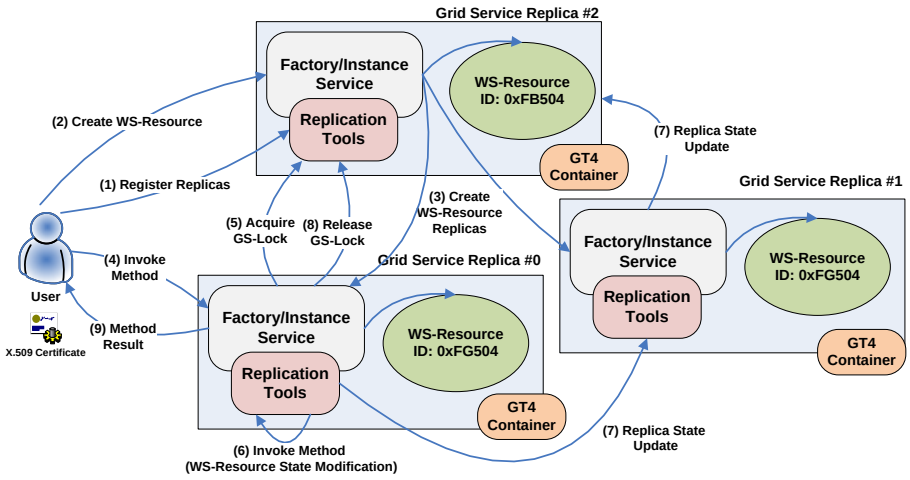

Fig. 1. Main functionality of the replication library, using a ring topology

The replication library, whose functionality is fully described in the paper by Moltó et al. [3], provides an existing WSRF-based Grid service with automatic replication of its WS-Resources in a transparent way to the users, integrated with its GSI-based security mechanisms. The library operates in a multi-primary passive replication scheme, that is, any replica can process a request and the state of the modified WS-Resource is transferred to the other replicas. A good introduction to WSRF and WS-Resources, as well as the Factory/Instance design pattern used in the replication library can be found in the book by Sotomayor et al. [8].

Figure 1 summarizes the main functionality of the replication library, where the replicas are configured in a ring topology. The topology affects the order in

which the state is propagated from the replica whose state is changed. The figure shows the same Grid service, which includes the functionality of the replication library, deployed on different GT4 containers. These containers are typically hosted by different machines, but this is not a requirement of the replication library, since different GT4 containers could be running within the same machine but listening on different ports.

According to the figure, first of all the user chooses any Grid service replica and registers the other replicas (step 1). Each replica can be uniquely identified by its URI (Uniform Resource Identifier). Next, the user requests this replica to create a WS-Resource (step 2), which stores the state of the Grid service. This WS-Resource is then created on the other replicas with exactly the same key and the same ownership (step 3). This requires credential delegation among the replicas and an appropriate configuration of client-side stubs to satisfy the GSI-based security of the Grid services. Sharing the same identifier simplifies the client-side access to the different replicas. In addition, all the replicas agree on a common topology, which in the case depicted is a ring topology. At the end of this process, the same WS-Resource exists on the different container instances. Therefore, the replication of the Grid service is achieved by a replication of its state, which is stored in its WS-Resources.

When an user invokes a method that results in a state modification (step 4), synchronization protocols among the replicas are required to maintain a consistent state. To ensure coordinated access to the state of any WS-Resource, the library implements a lock-based approach, via Grid services, called GS-Lock. Initially, all the replicas agree on the one with the largest index to be the GS-Lock controller. Before a Grid service executes an operation that modifies the WS-Resource, it must acquire the GS-Lock. This is a blocking operation that returns immediately if the GS-Lock is available but it hangs the caller until it becomes available and access is granted to it (step 5). This may result in a small client-side delay, but the client is completely unaware of the synchronization and replication process. This approach ensures sequential consistency, that is, the guarantee that all the replicas traverse the same state changes in the same order.

Once the GS-Lock has been acquired, the operation that modifies the state of the WS-Resource proceeds. When a state modification is detected, the replication library propagates the state update to the other replicas using a coordinated distributed algorithm that depends on the topology (step 7). Once all the replicas have been successfully updated, the GS-Lock is released (step 8) and the result of the method is returned to the user (step 9). This allows another operation to gain access to the GS-Lock and proceed with the WS-Resource state modification.

The replication library can use a ring-based topology, as depicted in the left side of Fig. 2. Whenever the user performs an operation on a replica that affects its state (i.e. the WS-Resource), the modified replica starts a distributed algorithm for state updating through all the ring. However, as the number of replicas grow, the amount of time dedicated to state propagation (the time required since a replica is modified until all the replicas share the same state)

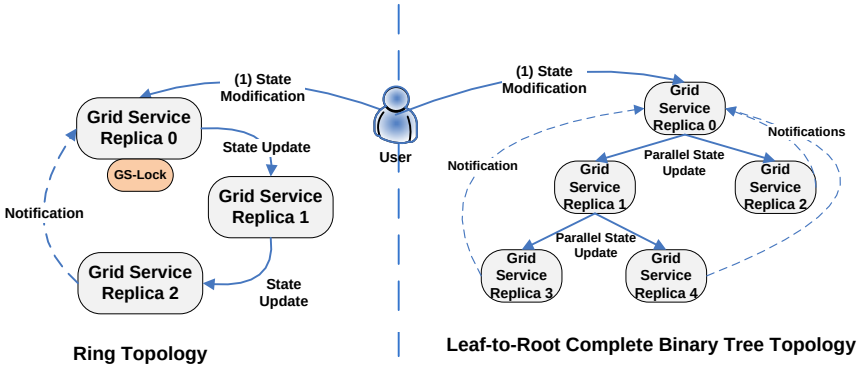

Fig. 2. Interaction diagram of the Grid service replication library. The left-hand side depicts a ring-based topology while the right-hand side shows a Leaf-to-Root Complete Binary Tree Topology.

increases linearly. This may impose serious limitations for replicated Grid services with a large number of replicas. In order to reduce this time, the replication library can use a Leaf-to-Root Complete Binary Tree topology, as depicted in the right side of Fig. 2. A Complete Binary Tree is a binary tree in which every level, except possibly the last, is filled, and the nodes in the last level are all to the left. This topology allows concurrent parallel update streams through the branches of the tree. Since a Complete Binary Tree is always perfectly balanced, this state update process requires a time in the order of $\theta(log_2(N))$ for a tree with N replicas, thus resulting in a significantly faster approach. Furthermore, since these kind of trees can be fully represented using an array, this allows designing very efficient distributed algorithms for state update. See [3] for the details about the distributed algorithms employed, the protocols for joining a group, the fault-tolerant capabilities of the library and its performance evaluation.

The usage of distributed machines allows to cope with failures in the Grid services while maintaining service availability. However, including new replicas requires configuring additional machines with the Globus Toolkit 4 and deploying the specific Grid Service to be replicated and its configuration. Requiring human intervention imposes a clear limitation on the scalability of this approach when many replicas fail. Enabling the replication library to automatically resize the group of replicas paves the way to dynamic on demand scalability of Grid services. Notice that there are several scenarios that require the dynamic provision of new Grid service replicas. On the one hand, maintaining service provision requires that new replicas substitute the failed ones, avoiding service disruption. On the other hand, it might be necessary to temporarily increase the number of replicas for Grid services experimenting a huge amount of queries by the users. This would be specially useful for Grid services exposing information, such as catalogs and information systems.

This is precisely the aim of this work, to integrate the replication library with a Cloud infrastructure to automatically deploy and configure Virtual Machines able to act as new Grid service replicas.

3 Cloud Computing Technologies Employed

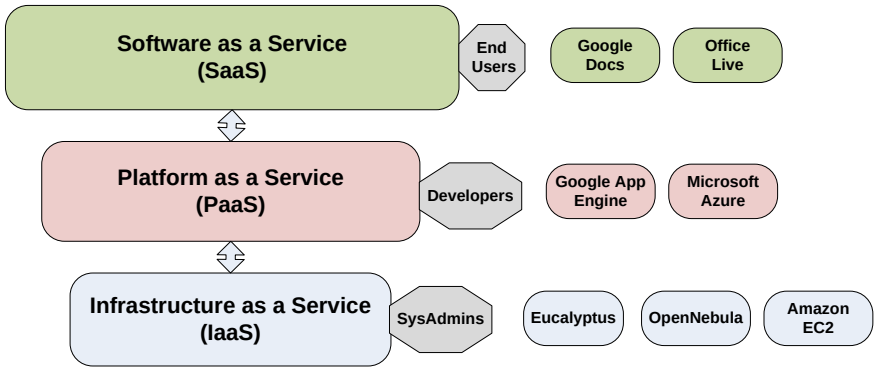

Fig. 3. A classification of Cloud-based approaches, with their target user community and samples.

Cloud approaches currently embrace a broad scope of technologies and applications. Figure 3 describes one of the classifications of systems denoted by the Cloud term. On the top level we have *Software as a Service* (or SaaS), where end users are provided with applications that are hosted online, such as Google Docs or Office Live. This way, application management no longer relies on the user, who can benefit from applications, and the required resources, on a pay-per-use basis (depending on the service). In the middle level, *Platform as a Service* (or PaaS) appears, which provides developers with a programming environment and APIs for creating Cloud-based applications that typically consume the computational resources of the platform. Examples of this approach are the Google App Engine [9] and Microsoft Azure [10]. Finally, the lowest level includes *Infrastructure as a Service* (or IaaS), where the usage of Virtual Machine Managers turn a physical computing infrastructure into an elastic computing platform based on Virtual Machines. Notice the interaction among the different levels, since a SaaS application can be developed using the framework provided by a PaaS, which in turn might harvest the computing power from an IaaS cloud.

In this particular work we focus on IaaS approaches which allow the dynamic deployment and management of Virtual Machines (VMs). This provides the foundations from which to contextualize the VMs into the required GT4-based Virtual Appliances (VAs) to be used as Grid service replicas.

3.1 Virtualization Technologies

Running a VM (with its guest OS) on top of a physical hardware (with its host OS) requires the usage of virtualization and hypervisor technologies installed on the host machine. There are currently different open source virtualization approaches. Xen [11] provides open source software for paravirtualization. With this approach, the guest operating system runs a modified operating system. Xen also supports full virtualization, which is the approach performed by KVM. KVM (Kernel-based Virtual Machine) [12] is a full virtualization solution for Linux that requires the CPU to support virtualization extensions (Intel VT or AMD-V). Under this approach, each VM within the same host runs as an independent process which has its own virtualized hardware. One of the benefits of full virtualization is to run virtual machines with an unmodified version of the guest operating system. With this approach, the hypervisor is responsible for giving the VM an illusion of a dedicate hardware machine.

We have relied on the KVM hypervisor. This software is gaining momentum as it is included in the Linux kernel since version 2.6.20. As we use an infrastructure based on recent Blade servers, with the required virtualization extensions, KVM is provided with the appropriate environment to deliver good performances.

3.2 Virtual Machine Managers

Virtual Machine Managers (VMMs) are responsible for allocating and monitoring the VMs into the existing hardware infrastructure. They typically take the VM definition along with its disk image and perform the required actions to get the VM up and running on top of a hypervisor.

There are currently different software and approaches for Virtual Machine Management. Eucalyptus [13] is an open source software for implementing cloud computing using the hardware infrastructure of organizations. It was designed to provide the functionality and APIs of Amazon EC2 [14], but for private clouds. It currently exists both as an open source projects and a commercial application. Nimbus [5] is an open source toolkit that allows to turn a cluster into an IaaS cloud. It was built using Grid services based on the Globus Toolkit 4. In addition, it can be integrated with schedulers like PBS or SGE and allows to deploy a group of VMs to create clusters on demand. It currently provides APIs based both on Amazon EC2 and on WSRF (Grid services). OpenNebula [15] is an open source virtual infrastructure manager which manages the storage, network and virtualization technologies to enable the dynamic placement of virtual machines on a Cloud infrastructure. It also allows the migration of VMs among the different resources of the infrastructure. AbiCloud [16] is the open source software for managing a Cloud infrastructure from the Abiquo enterprise. It offers the users a GUI to manage their VMs and supports the principal hypervisors. In addition, it supports industry standards such as the Open Virtualization Format [17].

For this particular work we have relied on OpenNebula. This software stands out for its ability to provide a gateway in order to supply application-dependent

contextualization information into the VM. This is achieved by allowing the users to specify the files that are required to be available inside the VM in a declarative manner. When a request is submitted to start an instance of the VM, OpenNebula creates a virtual disk that is attached to the VM at boot time. This way, the VM can access these files which can be used to properly contextualize the VM with application-dependent information.

4 Contextualization: From Applications to Scientific Virtual Appliances

The usage of VMMs allows the dynamic deployment of virtual machines on top of physical computational resources, supporting the generation of Cloud computing infrastructures. However, the execution of scientific applications in this kind of computational infrastructures requires the development of Virtual Appliances (VA). A VA consists of a VM that includes the application and the entire required environment for its execution (numerical libraries, databases, runtime environments, etc.). With this approach, the hardware infrastructure is decoupled from the applications, which are completely encapsulated and self-contained, thus guaranteeing a successful execution.

The process of configuring a VM to obtain a VA is referred to as contextualization. An application with a reduced number of external dependencies can be perfectly contextualized at the time the VM is deployed by the VMM. This way, it is possible to start from a base VM, that only includes the operating system and common use libraries, and to perform the application deployment and contextualization when the VM boots, before executing the application.

However, other applications have a large number of dependencies on external software components (such as libraries, data base management systems, etc.), and it is not feasible to perform the contextualization at the time of deployment. This is because, in some cases, the time needed for contextualization could exceed the time of running the application itself. 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 installing of the most complex software components by the user, in order to produce a pool of partially contextualized VMs. 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.

4.1 Contextualizing Scientific Virtual Appliances

We are working on a tool that enables to deploy scientific applications with minimal user intervention [18]. 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.

With the proposed approach, 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 is in charge of automatically performing the typical 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.

The contextualization tool offers a plugins-based approach so that application developers can declare the installation approaches required by their packages. This way, it is possible to chain different software installation descriptions in order to perform complex installations. Other software configuration tools exist such as Chef [19], but our approach requires an XML declarative description instead of writing Ruby code. In addition, it focuses on the specific workflow required for the deployment of scientific applications. 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 A Cloud back-end for the Grid Service Replication Library

This section describes how the replication library has been modified to include OpenNebula's capabilities to deploy new virtual machines and its coupling with the contextualization software in order to configure the VMs into new Grid service replicas based on GT4.

Concerning the creation of the Virtual Appliance (VA) we might start from a plain VM with just an OS and perform the contextualization at boot time in order to install GT4 and the Grid services of both the replication library and the specific Grid service to be replicated. However, the installation of GT4 requires several minutes and, therefore, it is unfeasible to do it at boot time.

Thus, we created a VA based on Ubuntu JeOS 8.04 to be used for the KVM hypervisor. A vanilla installation offers a small footprint (380 MBytes of disk space). Performing the installation of the full Globus Toolkit 4.0.8 inside the VM enlarges the disk image size up to 3.3 Gbytes. This size matters, as VMMs

typically copy this file into the physical resources in order to run separate instances of the VM. Therefore, in order to reduce this size, we created a cloned VM in which we performed the installation of the GT4, and later transferred the result of the compilation into a vanilla Ubuntu JeOS. This results in a 1.1 GBytes disk image size. We specially configured the VM so that the Grid services container automatically started at boot time. This results in a pre-contextualized VM that only requires the deployment of the specific Grid service and security configuration at boot time to be used as a new Grid service replica.

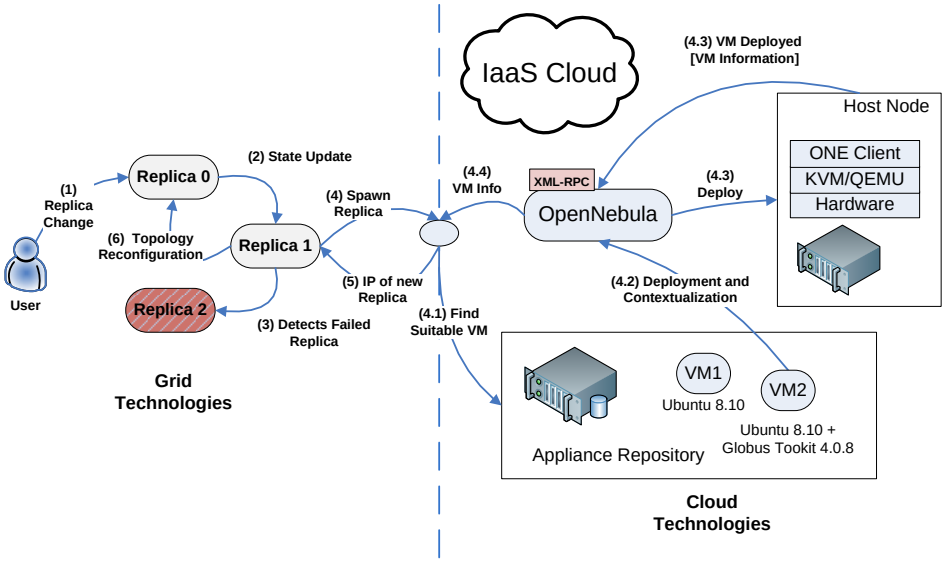

Fig. 4. Integration of the Grid service replication library and the Cloud computing back-end.

Figure 4 summarizes the main interaction diagram of the replication library when combined with the Cloud computing infrastructure. Notice that the figure is logically split in two sections. The left hand side only involves Grid technologies, i.e., the Globus Toolkit, WSRF-based Grid Services, X.509 proxies, certificates, etc. The right hand side focuses on Cloud technologies, i.e., virtualization (KVM), infrastructure management (OpenNebula) and Virtual Appliances which, by the way, internally use Grid technologies.

As depicted in Fig. 4, the user performs an operation on any WSRF-based Grid Service replica that modifies its WS-Resource state (step 1). This triggers a state update process that traverses the topology (a ring topology is shown for simplicity). When the replica 1 detects that replica 2 is unreachable (step 3) it decides that a new replica should be spawned (step 4).

To create a new Grid service replica, the appliance repository is queried for a valid Virtual Appliance (step 4.1). This repository is currently a Web service that enables the users to catalog their VMs and tag them so that they can be indexed and searched. In this particular case, we query for the availability of a VA based on Linux and with the Globus Toolkit 4.0.X installed to satisfy the requirements of the replication library. Next, OpenNebula deploys the VA on the best candidate host node, according to its internal scheduling policy. The description of the VA includes a template file which specifies the disk image, its memory and network requirements and the contextualization information. This includes a copy of the *.gar* files required to deploy the Grid service, the host certificate, and the contextualization tool required to perform the appropriate configuration of the virtual machine. All these files are packed into an ISO image which is attached as a disk into the VM by OpenNebula. This way, when the VM boots, the contextualization tool is started to deploy the Grid service and the required configuration.

Once the VM is up and running, OpenNebula can be queried to obtain information about the new instance. To programmatically use OpenNebula from the replication library, we take advantage of the XML-RPC interface offered by the OpenNebula server, which exposes almost a one-to-one mapping of all the functionality available via the command-line. This way, the replication library knows the IP of the new Grid service replica (step 5) and the topology reconfiguration process provided by the replication library can proceed (step 6). The final result is a new Grid service replica that was deployed on demand and which substitutes the failed one so that the group of replicas can maintain the work load. The state of the new replica is supplied by the replica that detected the failed one (see [3] for details).

For this proof-of-concept implementation, we currently assume that all the Grid service replicas have access to the contextualization information, just in case they have to trigger the creation of a new replica. However, we plan to introduce a contextualization service that centralizes the information concerning the different plugins employed to contextualize the applications.

6 Discussion and Deployment Time of Replicas

In order to test the described approach, we have recreated some replication scenarios in a Blade server composed of four dual-processor Intel Xeon Quad-Core with 16 GBytes of RAM. The machine is configured as a cluster and, therefore, one of the nodes acts as the frontend and does not receive the allocation of VMs. OpenNebula 1.4 is installed and specially configured to use the NFS transfer driver. This way, when a new virtual machine is deployed, OpenNebula performs a copy of the base image into the destination node (via NFS) and boots it.

Deploying a new replica on demand on the Cloud Computing infrastructure introduces the following overheads:

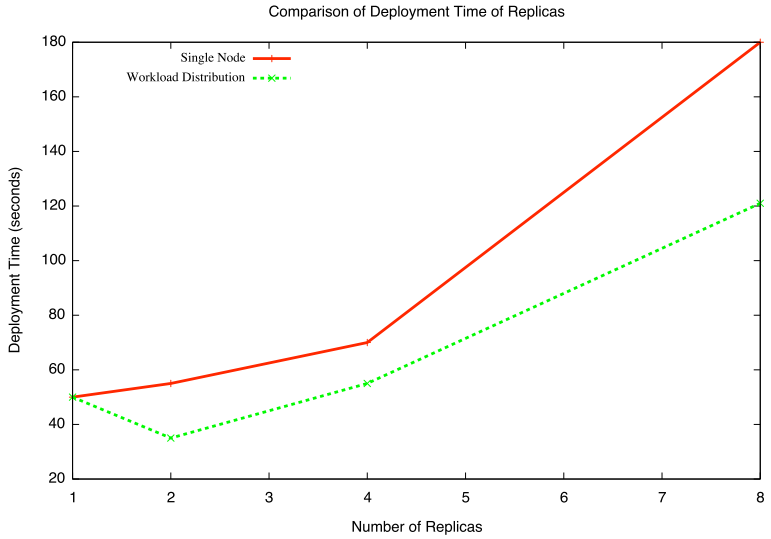

Fig. 5. Comparison of the deployment time for a number of replicas. The dotted line represents a round-robin allocation of virtual machines into three different computing nodes. The solid line represents all the replicas allocated to a single node.

1. Querying the Appliance Repository. We currently hold the Virtual Appliances Repository in the same machine where the OpenNebula server resides. At the moment, this operation represents a negligible cost.
2. Deployment of the Virtual Appliance. OpenNebula performs the copy of the VA image disk file into the host node so that different instances of the same VA behave as independent copies. Notice that this approach naturally fits with a cluster-based infrastructure where the front-end stores the VAs and the internal nodes provide computing support to run the VAs.
3. Contextualization of the VA. This requires deploying at boot time of the VA the latest version of the Grid service and required certificates. This step requires 5-6 seconds on average.

Therefore, since the contextualization time is very low, the size of the Virtual Appliance is crucial to reduce the overhead in deploying a new replica. Figure 5 summarizes the result of a comparison of the deployment time, in seconds, of a different number of replicas using just one node and three nodes of the Blade in order to achieve workload distribution. The deployment phase includes the time required since a request to the Virtual Machine Manager (OpenNebula) is performed until the VM is up and running. This comparison allows to know the minimum time required to obtain new Grid service replicas. The solid line shows the deployment time where all the virtual machines are allocated to a single node of the Blade server. The dotted line shows the results of a round-robin allocation of virtual machines in the three nodes of the Blade, in order

to distribute the virtualization overhead time among different machines. Since the VM image resides in the frontend, a copy must be performed into each node before booting. This introduces a bottleneck since all the nodes must perform this data transfer, although the influence of the cache of the NFS protocol can help reducing the transfer cost. Once each node has its own copy of the disk image, the booting and contextualization in each node performs concurrently without sharing resources with other nodes. Therefore, as it is expected, a workload distribution provides a reduced deployment time. To investigate the impact of this approach into the replication library, it is important to focus on the total replication time. According to the figure, it is possible to concurrently deploy 4 new replicas in slightly over a minute using three nodes.

It is important to point out that achieving high availability requires both replication and distribution. This proof-of-concept tests have been performed in a single infrastructure, what allows to measure the impact of using Cloud-based technologies. Therefore, the usage of separate Cloud infrastructures would be the ideal situation to distribute the different service replicas. In this situation, the data transfer cost of the pre-contextualized VM image would certainly be preponderant. For that, distribution of the disk image among the different providers, as close as possible to the computational infrastructure, would certainly be required to reduce the overall impact of the deployment time of new replicas.

7 Conclusions and Future Work

This paper has described the usage of a Cloud computing infrastructure by a WSRF-based Grid Service replication library in order to automatically deploy new replicas in the case of failure. For that, we have coupled the functionality of a Virtual Machine Manager for the dynamic allocation and monitoring of virtual machines, with a contextualization tool that eases the deployment of scientific applications into virtual machines. This approach allows to guarantee service provision by the automatic replacement of failed replicas.

Future works include the development of a software module into the replication library in order to dynamically scale the number of replicas required to provide appropriate service provision without involving excessive resource consumptions (deploying more replicas than the required ones). This would require the definition of upper and lower boundaries to the number of replicas, what is certainly application-dependent. In addition, we plan to extend our approach into large scale Cloud infrastructures in order to couple replication with distribution for increased high availability.

References

1. 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

2. Snelling, D., Robinson, I., Banks, T.: OASIS web services resource framework (WSRF) TC. Online (2006) http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.
3. 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
4. Vázquez, C., Huedo, E., Montero, R.S., Llorente, I.M.: Dynamic provision of computing resources from grid infrastructures and cloud providers. In: 2009 Workshops at the Grid and Pervasive Computing Conference. (2009) 113–120
5. Keahey, K., Freeman, T., Lauret, J., Olson, D.: Virtual workspaces for scientific applications. In: Proceedings of the SciDAC 2007 Conference. (2007)
6. Somasundaram, T.S., Amarnath, B.R., Kumar, R., Balakrishnan, P., Rajendar, K., Rajiv, R., Kannan, G., Britto, G.R., Mahendran, E., Madusudhanan, B.: CARE resource broker: A framework for scheduling and supporting virtual resource management. Future Generation Computer Systems **26**(3) (2010) 337 – 347
7. Kertesz, A., Kecskemeti, G., Brandic, I.: An SLA-based resource virtualization approach for on-demand service provision. In: VTDC '09: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing, New York, NY, USA, ACM (2009) 27–34
8. Sotomayor, B., Childers, L.: Globus Toolkit 4: Programming Java Services. Morgan Kaufmann (2005)
9. Google: Google app engine. code.google.com/appengine (2010)
10. Chappel, D.: Introducing windows azure. Technical report, Microsoft (2009) http://view.atdmt.com/action/mrtyou_FY10AzureWhitepaperIntroWindowsAzureSec_1.
11. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, New York, NY, USA, ACM (2003) 164–177
12. Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: KVM: The linux virtual machine monitor. In: OLS '07: The 2007 Ottawa Linux Symposium. (2007) 225–230
13. 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)
14. Amazon: Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2 (2010)
15. 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)
16. Abiquo: Abicloud. http://www.abiquo.com/ (2010)
17. (DMTF), D.M.T.F.: Open virtualization format 1.1. http://www.dmtf.org/standards/published_documents/DSP0243_1.1.0.pdf (2010)
18. Moltó, G., Hernández, V.: Management and contextualization of scientific virtual appliances. In: Cloud Futures 2010: Advancing Research with Cloud Computing. (2010)
19. Opscode: Chef. http://wiki.opscode.com/display/chef/ (2010)