Document downloaded from:

http://hdl.handle.net/10251/35846

This paper must be cited as:

Moltó, G.; Calatrava Arroyo, A.; Hernández García, V. (2013). A service-oriented architecture for scientific computing on cloud infrastructures. En High Performance Computing for Computational Science - VECPAR 2012. Springer Verlag (Germany). 163-176. doi:10.1007/978-3-642-38718-0_18.

The final publication is available at

http://link.springer.com/chapter/10.1007%2F978-3-642-38718-0_18

Copyright Springer Verlag (Germany)

A Service-Oriented Architecture for Scientific Computing on Cloud Infrastructures *

Germán Moltó[1], Amanda Calatrava[1], and Vicente Hernández[1]

Instituto de Instrumentación para Imagen Molecular (I3M). Centro mixto CSIC
Universitat Politècnica de València CIEMAT, camino de Vera s/n, 46022 Valencia, España
{gmolto,vhernand}@dsic.upv.es, amcaar@ei.upv.es

Abstract. This paper describes a service-oriented architecture that eases the process of scientific application deployment and execution in IaaS Clouds, with a focus on High Throughput Computing applications. The system integrates i) a catalogue and repository of Virtual Machine Images, ii) an application deployment and configuration tool, iii) a meta-scheduler for job execution management and monitoring. The developed system significantly reduces the time required to port a scientific application to these computational environments. This is exemplified by a case study with a computationally intensive protein design application on both a private Cloud and a hybrid three-level infrastructure (Grid, private and public Cloud).

Topics Parallel and Distributed Computing.

1 Introduction

With the advent of virtualization techniques, Virtual Machines (VM) represent a key technology to provide the appropriate execution environment for scientific applications. They are able to integrate the precise hardware configuration, operating system version, libraries, runtime environments, databases and the application itself in a Virtual Machine Image (VMI) which can be instantiated into one or several runnable entities commonly known as Virtual Appliances. With this approach, the hardware infrastructure is decoupled from the applications, which are completely encapsulated and self-contained. This has paved the way for Cloud computing [1, 2], which enables to dynamically provision and release computational resources on demand.

The efficient and coordinated execution of scientific applications on Cloud infrastructures requires, at least: (i) the dynamic provision and release of computational resources (ii) the configuration of VMs to offer the appropriate execution environment required by the applications and (iii) the allocation and

* The authors wish to thank the financial support received from the Generalitat Valenciana for the project GV/2012/076 and to the Ministerio de Economía y Competitividad for the project CodeCloud (TIN2010-17804).

execution of the jobs in the virtualised computational resources. This requires the coordination of different Cloud-enabling technologies in order to automate the workflow required to execute scientific application jobs on the Cloud. To that end, we envision a system where users express their application requirements via declarative procedures and the burden of its deployment, execution and monitoring on an IaaS (Infrastructure as a Service) Cloud is automated. There are previous studies that aim at using Cloud computing for scientific computing [3, 4]. However, as far as the authors are aware, there is currently no generic platform that provides automated deployment of scientific applications on IaaS Clouds which deals with VMI management, configuration of VMs and the meta-scheduling of jobs to the virtual computing resources. This represents the whole life cycle of scientific application execution on the Cloud.

For that, the main contribution of this paper is to present a service-oriented architecture integrated by the following developed components: i) a generic catalogue and repository system that indexes VMIs together with the appropriate metadata describing its contents (operating system, capabilities and applications), ii) a contextualization system that allows to deploy scientific applications together with its dependences, iii) a meta-scheduler to manage and monitor the execution of jobs inside VMs and to access the generated output data of the jobs with support for computational steering. The usage of such a system would significantly reduce the time required to migrate an application to be executed on the Cloud. The integration of the different components of the architecture enables to abstract many of the details that arise when interacting with Cloud platforms. This would reduce the entry barrier to incorporate the Cloud as a new source of computational power for scientific applications. This way, scientists would focus on the definition of the jobs and delegate on the proposed platform the orchestration of the components to execute the jobs on the provisioned virtualised infrastructure on the Cloud.

The remainder of the paper is structured as follows. First, section 2 introduces the architecture and details the features of the principal components. Later, section 3 addresses a case study for the execution of a protein design scientific application using the aforementioned system. Finally, section 5 summarises the paper and points to future work.

2 Architecture for Scientific Application Execution on the Cloud

Many scientific applications require the execution of batch jobs, where each job basically consists of an executable file that processes some input files (or command line arguments) and produces a set of files (or data to the standard output) without the user intervention. This is the case of many parameter sweep studies and Bag of Tasks (BoT) applications commonly found in High Throughput Computing (HTC) approaches, where the jobs share common requirements. For these applications, the benefits of the Cloud are two-fold. Firstly, computational resources can be provisioned on demand according to the number of jobs to be executed (and the budget of the user in the case of a public Cloud). Secondly, the provisioned VMs can be configured for the precise hardware and software configuration required by the jobs. This means that VMs can be reused to perform the execution of multiple jobs.

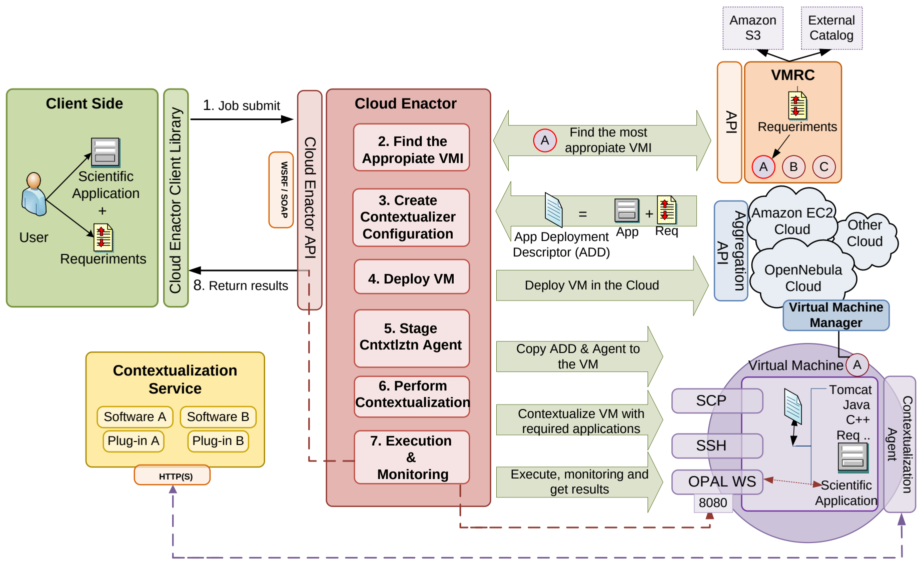


Fig. 1. Scientific applications execution on IaaS Cloud via the Cloud Enactor

Figure 1 summarises the main interactions between a user and the proposed architecture. The user employs the client-side API to describe each task to be executed (executable file or source code, and required input files) together with the hardware (i.e. CPU architecture, RAM, etc.) and software requirements (OS, applications, system packages, etc.). The jobs might optionally include budget information, since the underlying Cloud infrastructure could require a pay-per-use access to resources. These jobs are submitted (step 1) to the Cloud Enactor (CE) which is the central manager that orchestrates all the components.

The CE checks whether the job could be executed on one of the already deployed (if any) VMs. For the jobs that cannot be executed on the currently deployed VMs, the CE queries the Virtual Machine image & Repository Catalogue (VMRC) [5] with the job's requirements to find the most appropriate VMI to execute the application (step 2). The VMRC, a software that we previously developed, implements matchmaking capabilities to offer a ranked list of suitable VMIs to the Cloud Enactor. The VMRC discards the VMs that do not satisfy the mandatory requirements (i.e., different OS or CPU architecture) and it ranks the resulting VMIs according to the degree of satisfaction with respect to the optional requirements (mainly, software applications). The CE computes the deviation from the current state of the most appropriate VMI found and the desired state for the job execution in order to create the Application Deployment Descriptor (ADD) for the contextualization software (step 3). The ADD specifies the deployment process of the application so that the contextualization software can unattendedly perform the installation of the application and its software dependences. This will be executed inside the VM at boot time to deploy the application and its dependences.

Next, the CE must decide the deployment strategy of VMs, which will be in charge of executing the jobs. For that, it has to consider a mixture of performance, economic and trust models to decide the optimum number of VMs to be deployed, together with their Cloud allocation strategy. The performance model should consider the execution time of the jobs (which can be initially estimated by the user but computed after each execution), the deployment time of the VM in the Cloud infrastructure, the time invested in deploying the software requirements of the job (contextualization) and the application itself, as well as the time invested in data transfer, that is, staging out the generated output data of the application inside the VM. The economical model should consider the budget of the user allocated to the execution of each job (or a set of jobs), and the billing policies of the Cloud provider (i.e. hourly rates, economic time zones, etc.). Finally, the trust model plays an important role on scenarios with multiple Cloud providers (Sky Computing), where reputation and the ability of a provider to systematically fulfill the Service Level Agreement (SLA) must be considered. The trust model would be employed to rank a Cloud provider according to its adherence to SLA and the Quality of the Service it offered along the time, among other possible characteristics. For example, a Cloud provider that systematically violates its own SLA should be ranked lower than a provider that has always fulfilled the terms of conditions. The user would express the precise rank function according to the aforementioned categories, as performed in other meta-scheduling softwares such as GridWay.

Therefore, the CE decides to fire up a new VM (or a group of them). This is achieved by delegating on a Virtual Infrastructure Manager (VIM), which deploys the VM on top of a physical infrastructure (step 4). Notice that the CE could use elasticity rules in order to enlarge or shorten the number of VMs dynamically assigned for the allocation of jobs, depending on the budget and the deadline constraints imposed by the user.

When the VM has booted, the CE stages the contextualization agent and the ADD into the VM using SSH (step 5). The VMRC service stores the login name and the private key (or the password) of an account in the VM as part of the metadata stored for a VMI. Then, the contextualization process is started, where software dependences are retrieved from the Contextualization Service and then installed. Next, the scientific application is deployed and a Web services (WS) wrapper is automatically created and deployed into an application server, which is finally started (step 6). This WS wrapper enables to remotely start and monitor the application running inside the VM. All this automated process results in a VA fully configured for the execution of the scientific application.

Once the VA is up and running, the meta-scheduler can perform the execution of the jobs inside the VAs (step 7). This involves managing and monitoring the execution of the jobs inside the VM during their lifetime. For efficiency purposes each VM would be in charge of the execution of several jobs. In the case of parameter sweep studies and BoT applications commonly found in HTC approaches, the jobs share common requirements and, therefore, they can be executed in the same contextualized VM. In addition, scientific applications might require a periodical access of the generated output data during their executions, mainly for computational steering purposes. Once the application inside the VM has finished executing, then its output data must be retrieved so that another job (with the same requirements) can execute inside the VM.

After all the executions have been carried out, the VAs can be gracefully shutdown which is achieved by the VIM. Notice that it is possible to catalogue the resulting VMI (after the contextualization process) together with the metadata information concerning the new applications installed. Therefore, this would minimize the contextualization time for subsequent executions of that scientific application, since no additional software should have to be installed. This streamlined orchestration of components enables the user to simply focus on the definition of the jobs and thus delegate to the central manager the underlying details of interacting with the Cloud technologies for computational resource provisioning and scientific application execution.

This Service-Oriented Architecture relies on several interoperable services that can be orchestrated by the Cloud Enactor due to the usage of standard protocols and interfaces (WS, WSRF, XML). Concerning the software employed, we have relied on the GMarte meta-scheduler [6], which provides execution management capabilities of scientific tasks on computational Grid infrastructures. By incorporating the functionality to access Cloud infrastructures in this software we can simultaneously schedule jobs on both Grid and Cloud infrastructures. In fact, once the virtual infrastructure of computational resources has been provisioned, other job dispatchers could be fit within the proposed architecture, such as Condor or GridWay. The WS Wrapper for the application is created by the Opal 2 Toolkit [7], which has been integrated in the lightweight contextualization software that we previously developed. Other tools for software configuration, such as Puppet or Chef could also be employed within this architecture.

2.1 The Virtual Machine Catalogue and Repository

In a previous work we introduced an early version of the Virtual Machine Catalogue and Repository system (VMRC) [5], whose main capabilities are explained in this section for the sake of completeness. This paper also describes novel features recently included in the system, such as multi-user support by means of Access Control Lists (ACLs) in order to introduce certain levels of security and prevent malware distribution in the VMs and the development of a web-based

GUI. In addition, the integration with the Cloud Enactor module is unique in this paper, which can be seen as a practical usage of its functionality.

The main goal of VMRC is to enable users to upload, store and catalogue their VMIs so that they can be indexed. This way, others can search and retrieve them, thus leveraging sharing and collaboration. For that, we have used industry standards such as the Open Virtualization Format (OVF) [8] to describe the VMIs in an hypervisor-agnostic manner, and Web Services to develop the core of the VMRC service.

Each VMI can be catalogued together with appropriate metadata including information about the hardware configuration (memory, architecture, disk size, etc.), the operating system (type of OS, version and release) and the applications currently installed (application name and version). Linking metadata to the VMI enables the development of matchmaking algorithms to retrieve the most appropriate VMI to execute a job considering its requirements.

The following snippet of code summarises the declarative language employed to query the catalogue considering the job's requirements. This example queries the catalogue for a Linux-based VMI, preferably an Ubuntu 11.10 or greater, created for the KVM hypervisor which must have MySQL 5.0 and Tomcat 7.0.22 and it would be desirable to have also the Java Development Kit version 1.5 or greater.

```
vm.type="kvm"
os.name="linux"
os.name="linux" && os.flavour="ubuntu" &&
         os.version>="11.10", soft, 20
app.name="org.mysql" && app.version="5.0"
app.name="org.apache.tomcat" && app.version="7.0.22"
app.name="org.oracle.java-jdk" &&
          app.version>="1.5", soft, 40
```

This language, inspired by the Condor classads language [9], differentiates between the hard requirements, which should be met by a VMI to be considered a potential candidate, and the soft ones, which can be ranked by the client. Certain applications might be considered soft requirements since the client might rely on proper deployment software to delegate the installation of these software on the VM. The inclusion of matchmaking capabilities in the catalogue is a key differential aspect with other catalogues of VMIs.

It is important to point out that the usage of preconfigured VMIs as base images for other VMIs involves security concerns that should be addressed, such as the distribution of malware among images. This can be alleviated by enforcing access control to images [10]. Therefore, we have included multi-user support in the VMRC. The VMRC has an administrator account that has privileges to create new users. The user that registers a VMI can optionally specify the list of users (or give public access) that can perform a given operation on its VMI (search, download, modify). This allows having public images in the catalogue, downloadable by everyone, and private images which might be shared by a collection of users. This is of importance for a research collaboration that might require the usage of a set of VMI, with their specific requirements for their scientific applications.

The VMRC features a web-based GUI which enables authentication via user and password in order to list and download the VMIs together with its metadata that the user can access. Therefore, the catalogue can currently be used via its Web Service API, through the Java bindings for programmatic access, and also using the web based GUI. This allows seamless access to the VMIs. Notice that since the VMRC is a generic component, and it has no specific bindings with a particular Cloud infrastructure, it can be deployed as a central VMI sharing module in a Cloud deployment in order to foster sharing and collaboration. This software is open source and it is available online[1].

2.2 Contextualization of Scientific Virtual Appliances

As stated earlier, the process of configuring a VM to obtain a VA can be referred to as contextualization, a term initially employed in [11] for the configuration of virtual machines to create virtual clusters. This term is employed in this paper for application contextualization, i.e., providing the application with the appropriate execution environment (mainly software dependences) to guarantee its execution. An application with a reduced number of external dependencies can be perfectly contextualized at the time the VM is deployed by the VIM. 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, applications with a large number of dependencies on third-party software are not candidate to perform the contextualization at the time of deployment. In some cases, the time required for contextualization might represent an important overhead, depending on the total execution time of the applications running on the VA. As an example, the compilation and installation of the Globus Toolkit 4 [12], a toolkit for deploying Grid services can take several hours. Additionally, in most cases, performing automatic contextualization requires a considerable complexity from a technical point of view. For these cases, a practical approach consists in performing the installation of the most complex software components by the user, in order to produce a pool of partially contextualized VMIs which are stored on the VMRC. 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.

Automatic Deployment of Scientific Applications In order to avoid manual installation procedures when the VMs are allocated by the VIM, we developed a tool (called *cntxtlzr*) that enables to automate the flow of deploying scientific applications. The main goal is to perform the main steps required when deploying a scientific application (packaging, configuration, compilation, execution) without the user intervention. This way, instead of manually configuring

[1] http://www.grycap.upv.es/vmrc

the VM via SSH, application inoculation into the VM with minimal user intervention is achieved.

This tool supports a small declarative language based on XML employed to create an Application Deployment Descriptor (ADD) which specifies the common actions employed to deploy a scientific application, together with its software requirements.

These are the typical steps involved in the deployment of a scientific application which are addressed by the developed tool:

1. Package Installation. Installs the software packages that the application depends on. It resolves dependencies with other software components and installs those dependencies first. The software packages can be made accessible to the contextualization software via an URL, an installable system package via *yum* or *apt-get* or simply staged into the VM together with the contextualization tool.
2. Configuration. Enables the user to detail the configuration process of the software package. This is achieved by specifying common actions such as copying files, changing properties in configuration files, declaring environment variables, etc.
3. Build. Compiles the software package using the appropriate build system (Configure + Make, Apache Ant, SCons, etc.)
4. Opal-ize. Creates the configuration required by the Opal toolkit, the Web services wrapper for the application. It then installs Opal and its requirements (Tomcat + Java), deploys the scientific application and, finally, starts Tomcat. This causes the scientific application to be deployed and the jobs ready to be started by the Cloud Enactor.

The following snippet of code shows a simplified version of an ADD. It describes an application called *gBiObj* that requires the MPICH Message-Passing Interface (MPI) library, the GNU C compiler and the *make* utility. Its source code is available in a compressed TAR file called *gBiObj.tgz*. We want the application to be accessible via the Opal WS Wrapper so we specify the Opalize XML element. In addition, we want to modify the Makefile of the scientific application to point to where the MPICH library has been installed. Notice that dependencies are installed before the application.

```
<DeployableApp name="gBiObj" requires="mpich gcc make">
  <Package name="gBiObj" file="gBiObj.tgz"/>
  <Opalize exec_file="gBiObj"
           default_args="--gra1 @gBiObj#INSTALL_PATH@/energy.gra"
<Configuration>
 <ReplaceInFile file="@gBiObj#INSTALL_PATH@/Makefile"
                from="mpicc" to="@mpich#INSTALL_PATH@/bin/mpicc"/>
</Configuration>
  <Build type="make"/>
</DeployableApp>
```

The contextualization tool relies on plugins, in the shape of other ADDs, to deploy specific software. This way application developers can specify the installation procedure required by their applications. Thus, it is possible to integrate different software installation descriptions in order to perform complex installations. There currently exists plugins for commonly used software such as Java, Globus Toolkit 4 WS-Core, etc.

The *cntxtlzr* tool currently consists of a highly portable Python script that processes the XML ADD and performs the required actions. This script is staged into the VM and started so that the contextualization process starts. The plugins (or ADDs) and the packages for the software dependencies can be stored in a separate web server. Therefore, the tool can download all the required information at runtime inside the VM in order to perform the contextualization. This lightweight approach to application contextualization only requires Python support in the VM, which is commonly found in the pristine installations of many GNU/Linux distribution.

We plan to combine our tool with other software configuration tools such as Puppet or Chef in order to take advantage of their software deployment approaches. Our approach would complement these software since we use a high level XML-based declarative description of the deployment process which targets at the specific workflow required for the deployment of scientific applications.

2.3 Application Management and Monitoring inside the VM

Starting and monitoring the execution of the jobs inside the VMs is far from being a trivial task because it requires the deployment of a special agent inside the VM in charge of starting and cancelling the application, and which provides information about the appropriate states of the job (running, finished, etc.). For that, we have relied on the Opal 2 Toolkit [7], which is a tool that wraps scientific applications as Web services so that they can be managed via remote invocations.

Opal requires the user to write an Application Configuration File (ACF) which provides metadata information about the application, such as the location of the executable file and the command-line arguments together with its description. It also accepts advanced features such as the execution method (either locally, inside the VM or delegating the execution to another component such as Globus or Condor).

Then, Opal generates a Web service wrapper and deploys the application into an application server such as Apache Tomcat. The WS front-end to the application allows starting, stopping and monitoring the application that runs in the VM. Different executions of the application can be concurrently carried out within the same VM, since separate folders are employed to generate output data files. It also allows to obtain a list of generated output files. An interesting point is that the output files can easily be accessed from outside the VM via the HTTP protocol, since they are generated inside the Tomcat deployment folder. This allows for computational steering capabilities, where scientific applications performing long simulations periodically generate output data. These data can be retrieved while the computation takes place, thus being able to steer the execution depending on the intermediate results. As an example, if a certain job in a Bag of Tasks submission takes longer than the expected time, it can be cancelled and resubmitted by the Cloud Enactor.

3 Case Study

In order to test the suitability of the Cloud infrastructure as a computational source for scientific applications, two case studies were performed. They involve a scientific application that designs proteins with targeted properties via a computationally intensive process based on Monte Carlo Simulated Annealing (MCSA) [13]. The application is developed in the C programming language and it depends on common build tools available in Linux (configure, make and a C compiler). It also requires the MPICH 2 library.

For the first case study, we used a fixed number of 8 jobs (an appropriate number for our test infrastructure) and we analysed the total execution time. This time includes from the beginning of the task allocation process until the last job has been executed and its output results have been retrieved. Each job requires the initial configuration of the protein and the matrix that indicates the energetic interactions among the different rotamers of the protein. This amounts to a total of 172 MBytes per job. The job outputs the results of the optimization process to the standard output. This computationally intensive application is typically CPU-bound, but we configured the executions to periodically read the energy matrix from the disk (as part of the optimization process) so that I/O would also be significant in the total runtime.

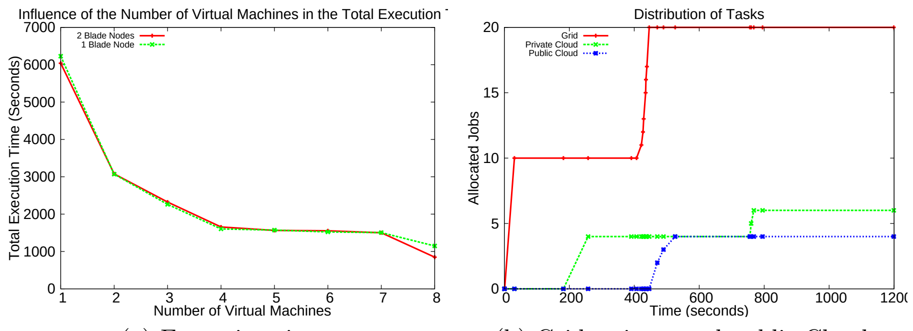
The test infrastructure is based on four dual-processor Intel Xeon QuadCore with 16 GBytes of RAM Blade servers, with a total of 32 cores, managed by OpenNebula 2.2 and the KVM hypervisor. Two nodes were exclusively used for this particular case study. In order to focus on the execution time, the case study was carried out on pre-started VMs where all the contextualization process had finished and the VMs were ready to receive the execution of the jobs. The allocation of tasks to VMs is achieved by the GMarte meta-scheduler. The current configuration controls that only one job is executed inside a single VM. Therefore, using N VMs allows the concurrent execution of up to N tasks. Other jobs are executed as soon as free VMs are available.

In addition, since the architecture can simultaneously schedule jobs to Grid and both private and public Cloud infrastructures, the second case study executes 30 protein design jobs on a hybrid infrastructure composed by resources from a Grid, the aforementioned private Cloud and the Amazon EC2 public Cloud. This demonstrates its ability to scale out computations on demand as long as resources from different infrastructures become exhausted.

3.1 Results

The solid line in Figure 2.a depicts the global execution time of the first case study. As expected, the global execution time decreases when the number of

VMs increases, since more computational resources are available to carry out jobs. The plateau in the execution time seen between 4 and 7 VMs is explained by the fact that only one job is executed in each VM and the execution time of each job is expected to be quite similar. Therefore, the executions are actually carried out in groups. As an example, with 5 VMs there is a first group of 5 jobs that are concurrently executed. When they finish, the meta-scheduler allocates the remaining 3 jobs to the free VMs. This would take a similar time as the allocation of the 8 jobs into 7 VMs, which carries out 7 concurrent jobs and a final single job when spare computational resources are available.



(a) Execution time (b) Grid, private and public Cloud

Fig. 2. Global execution time (a) of the case study, considering two different distributions of VMs. Allocated jobs (b) on an infrastructure composed of Grid, private Cloud and public Cloud resources.

The dotted line in Figure 2.a compares the degree of scalability of the Blade servers since it shows the global execution time of the case study when all the VMs are running inside a single node. It can be seen that a similar execution time is achieved except for the case of using 8 VMs, where a minor difference is noticed. Since each node features a dual quad-core processor, it appears that scalability issues are only noticeable starting from the 8-th VM in a single node, where the usage of shared resources such as memory and disk start affecting the execution of the applications. These results suggest that VM consolidation in few physical nodes might still deliver good performances for computationally intensive applications, depending on resource consumption.

Concerning the performance improvement gained using the Cloud infrastructure, the results show that up to an speed up of 7.13 is achieved with 8 VMs evenly distributed among the two physical nodes. The global execution time of the case study reduces from a total 6041 seconds in a single VM to just 847 seconds using the aforementioned 8 VMs. Therefore, the usage of virtualised resources from a Cloud as a provider of computational power can deliver a significant improvement for resource-starved scientific applications.

For the second case study we used 10 Grid nodes (from a local resource integrated in the Spanish National Grid Initiative), 4 provisioned VMs for the private Cloud and 4 for the public Cloud. The provisioned VMs where contextualized at boot time in order to deploy the application. We used the Free Usage Tier provided by Amazon EC2 to provision low-performance VMs, thus requiring a noticeably larger time to execute the jobs

The task allocation of the 30 jobs is shown on Figure 2.b, further detailed in [14]. The system starts submitting jobs to the Grid infrastructure until all the execution slots are used (approximately at instant 31 in the figure). Since there are pending jobs to be executed, a virtual infrastructure composed of 4 virtual machines is provisioned from the private Cloud provider in order to be able to submit additional jobs to be executed. When both the Grid infrastructure and the private Cloud are not able to execute additional jobs (approximately at instant 258 in the figure) then the computations are scaled out to the Amazon EC2 public Cloud provider. Therefore, 4 additional virtual machines on a pay-per-use basis are provisioned in order to enlarge the available computational infrastructure. Notice that from that moment on, the jobs are being concurrently executed on a Grid infrastructure and on virtual infrastructures provisioned from both a private Cloud and a public Cloud. When the provisioned computational resources of the Cloud are no longer used, they will be shut down. This enables to dynamically adjust the size of the virtual infrastructure to the computational requirements of the case study.

Therefore, the developed system allows to simultaneously harvest computational power from three different infrastructures, in order to reduce the execution time of HTC-based applications.

4 Related work

This paper aims at abstracting the details of scientific applications execution on Cloud platforms. The literature reveals research efforts into this area.

In a work related to the Nimbus project [15], the authors offer the Workspace Service, which enables to publish different VMIs ready to be used for the execution of certain applications. Therefore, each VMI must be properly configured in advance with the hardware parameters and software dependencies required for the execution of the application. A different approach is offered by the Swarm project [16] which is a task scheduler that acts over three kind of infrastructures (Grid, Windows Server Cluster and Cloud). However, the task execution on the Cloud requires the VMs deployed in the Cloud configured by means of a Hadoop cluster. It uses the MapReduce execution model for the execution of tasks.

SAGA [17] allows to remotely execute applications on top of Grid and Cloud infrastructures. The SAGA libraries and its dependences need to be deployed in advance into the VM, but the main advantage over the previous approaches is that it allows basic VM contextualization once it has been deployed in the Cloud. This includes package installation and minor application configuration during VM startup. There also exists the Cloud Scheduler[2], which is a cloud-enabled distributed resource manager. It provides part of the functionality of a VIM but uses the Condor scheduler [18] to delegate the scheduling decisions for jobs. The user can reference VMIs stored either in Nimbus (via its URL) or Amazon EC2 (via the name of the Amazon Machine Image (AMI)), the same IaaS providers currently available with this tool.

In [19] the authors propose a system to deploy and invoke science applications in the Cloud with minimal user effort. They address the principal challenges when porting an application to the Cloud: application deployment, application execution and data transfer from and into the Cloud. They propose several predefined application runtime environments which can be staged into the VM, and an execution framework to start the application. However, being implemented in Windows Azure [20], their approach only targets Windows platforms. In addition, their approach focuses on self-contained applications (binaries and libraries), which are assumed to seamlessly run on the target VM. Therefore, they do not consider the intricacies of deploying complex scientific applications.

5 Conclusion

This paper has introduced a software architecture that abstracts the details of application deployment and execution on IaaS Clouds. The system features the provision of computational virtualised resources, the configuration of these resources to support the execution of the applications, the cataloguing of virtual machine images and, finally, the job execution management on the virtual infrastructure. The benefits of the proposed architecture have been exemplified by the execution of a protein design case study on both a private Cloud infrastructure and a hybrid infrastructure (Grid, private and public Cloud). The automated deployment and execution of scientific applications fosters the widespread adoption of Cloud technologies by the scientific community. This way, Clouds deliver important benefits for scientific computing in terms of the ability to provision computational resources and the customizability of the execution environments.

Therefore, the main contribution of this work to the state-of-the-art is the development of generic components and an architecture to integrate them all in order to ease the process of executing scientific applications on the Cloud. In addition, some of the components of the architecture, such as the VMRC system, have been released to the community as open source.

References

1. Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M.: A break in the clouds. ACM SIGCOMM Computer Communication Review **39**(1) (2008) 50
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A.: Above the clouds: A berkeley view of cloud computing. Technical report, UC Berkeley Reliable Adaptive Distributed Systems Laboratory (2009)

[2] http://www.cloudscheduler.org

3. Rehr, J., Vila, F., Gardner, J., Svec, L., Prange, M.: Scientific computing in the cloud. Computing in Science **99** (2010)
4. 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)
5. Carrión, J.V., Moltó, G., De Alfonso, C., Caballer, M., Hernández, V.: A Generic Catalog and Repository Service for Virtual Machine Images. In: 2nd International ICST Conference on Cloud Computing (CloudComp 2010). (2010)
6. Moltó, G., Hernández, V., Alonso, J.: A service-oriented WSRF-based architecture for metascheduling on computational Grids. Future Generation Computer Systems **24**(4) (2008) 317–328
7. Krishnan, S., Clementi, L., Ren, J., Papadopoulos, P., Li, W.: Design and Evaluation of Opal2: A Toolkit for Scientific Software as a Service. In: 2009 IEEE Congress on Services. (2009)
8. Distributed Management Task Force (DMTF): The Open Virtualization Format Specification. (Technical report)
9. 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
10. Wei, J., Zhang, X., Ammons, G., Bala, V., Ning, P.: Managing security of virtual machine images in a cloud environment. ACM Press, New York, New York, USA (2009)
11. Keahey, K., Freeman, T.: Contextualization: Providing One-Click Virtual Clusters. In: Fourth IEEE International Conference on eScience. (2008) 301–308
12. Foster, I.: Globus toolkit version 4: Software for service-oriented systems. Journal of Computer Science and Technology **3779** (2006) 2–13
13. Moltó, G., Suárez, M., Tortosa, P., Alonso, J.M., Hernández, V., Jaramillo, A.: Protein design based on parallel dimensional reduction. Journal of chemical information and modeling **49**(5) (2009) 1261–71
14. Calatrava, A. In: Use of Grid and Cloud Hybrid Infrastructures for Scientific Computing (M.Sc. Thesis in Spanish), Universitat Politècnica de València (2012)
15. Keahey, K., Freeman, T., Lauret, J., Olson, D.: Virtual workspaces for scientific applications. Journal of Physics: Conference Series **78**(1) (2007) 012038
16. Pallickara, S., Pierce, M., Dong, Q., Kong, C.: Enabling Large Scale Scientific Computations for Expressed Sequence Tag Sequencing over Grid and Cloud Computing Clusters. In: Eigth International Conference on Parallel Processing and Applied Mathematics (PPAM 2009), Citeseer (2009)
17. Merzky, A., Stamou, K., Jha, S.: Application Level Interoperability between Clouds and Grids. 2009 Workshops at the Grid and Pervasive Computing Conference (2009) 143–150
18. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the Condor experience. Concurrency and Computation: Practice and Experience **17**(2-4) (2005) 323–356
19. Simmhan, Y., van Ingen, C., Subramanian, G., Li, J.: Bridging the Gap between Desktop and the Cloud for eScience Applications. In: 2010 IEEE 3rd International Conference on Cloud Computing, IEEE (2010) 474–481
20. Chappell, D.: Introducing windows azure. Technical report (2009)