Towards SLA-driven Management of Cloud Infrastructures to Elastically Execute Scientific Applications

Miguel Caballer**, Andrés García, Germán Moltó, and Carlos de Alfonso

Instituto de Instrumentación para Imagen Molecular (I3M).
Centro mixto CSIC - Universitat Politècnica de València - CIEMAT
Camino de Vera s/n, 46022 Valencia, España

Abstract. This paper summarizes the works towards a Service Oriented Architecture to abstract the execution of scientific applications under different programming models, with a special focus on High Throughput Computing. The platform features SLA-aware capabilities based on WS-Agreements and the ability to deploy customized virtual infrastructures with support for different IaaS Cloud providers. The proposed architecture features both horizontal (scale in/scale out) and vertical (scale up/scale down) elasticity capabilities in order to automatically fit the virtual infrastructure to the dynamic computing requirements of scientific applications. An overview of the platform is described, together with the current implementation state and issues to be considered.

1 Introduction

In the last years, many computing and data management infrastructures and technologies have been developed with the aim of providing resources in a scalable, transparent and reliable way, as any other utility. E-Infrastructures such as Grids have played an important role on integrating, sharing and exploiting distributed environments, and large experience has been acquired in many research groups. However, this model of IT resources as utility has not been really available until the development of Clouds. There is a great opportunity in both science and industry to use more efficiently computing infrastructures, reducing costs, production cycles, risk and environment impact. However, it still takes considerable effort to adapt current applications to Cloud environments.

Cloud infrastructures offer resources for computing and storage in the form of services, providing the flexibility, scalability and high availability needed by many scientific applications. The execution of applications in this kind of infrastructures is made by means of Virtual Appliances (VA), consisting of a Virtual Machine (VM) that includes the application and the entire environment required for its execution (numerical libraries, databases, runtime environments, etc.).

It is important to ease the development and deployment of applications in Cloud environments. In order to achieve this, suitable services and models have

** e-mail of corresponding author: micafer1@upv.es to be provided so that the effort required to successfully adapt applications to Cloud environments can be drastically reduced. Services will ease the management, contextualization and execution of the resources and applications, and models will enable applications to exploit the functionalities exposed. This means that many existing applications will benefit from the resources offered by cloud environments, while researchers will not need to spend time learning how to run their applications in a Cloud.

To that end, this paper outlines an architecture that aims at abstracting the execution details of scientific applications under different programming models on Cloud infrastructures with support to Service Level Agreements (SLA) features in order to guarantee the level of quality of service delivered to the application. The architecture aims at managing both horizontal and vertical elasticity in order to fit the underlying virtual infrastructure to the requirements of the application.

The remainder of the paper is structured as follows. First, section 2 outlines the general architecture proposed. Next, section 3 introduces the management of SLAs in order to guarantee the proper allocation of resources and the continuous fulfillment of the agreement during the execution of the application. Then, section 4 covers the approaches considered to manage elasticity both horizontally (increasing and decreasing the number of resources) and vertically (increasing and decreasing the capacities of resources). Finally, section 6 summarizes the paper and points to future works.

2 General Architecture to Abstract Execution

Figure 1 summarizes the main architecture employed to abstract the execution of scientific jobs on a Cloud under different programming models. Support for different programming models has been considered to be included in the platform (in particular MapReduce, Workflow, Master/Slave and MPI). In particular, this paper focuses on Master/Slave in the shape of the High Throughput Computing (HTC) paradigm implemented by means of PBS/Torque or by other related approaches such as Condor.

According to the figure, first of all the user has to describe the jobs to be executed and the requirements of the virtual infrastructure required to support the execution of the jobs (step 1). The jobs are described via a domain specific declarative language based on XML that declares the main properties of a job (executable file, input files, output files, computational requirements, elasticity rules, etc.). Most properties are common to different programming models but for High Throughput Computing (HTC) the scheme of independent jobs is commonly assumed. The infrastructure is defined by means of the Resource Application Description Language (RADL) language, a declarative language that specifies the desired capabilities of the virtual infrastructure, which has been employed in previous works [1]. The RADL aims at describing at a higher level the VM infrastructure that a user needs for a specific task. Concerning the description of the jobs, in case local files are referenced, these are copied to a Cloud Storage that supports the Cloud Data Management Interface (CDMI) standard (step 2). The usage of CDMI provides a uniform interface to store and access files regardless of

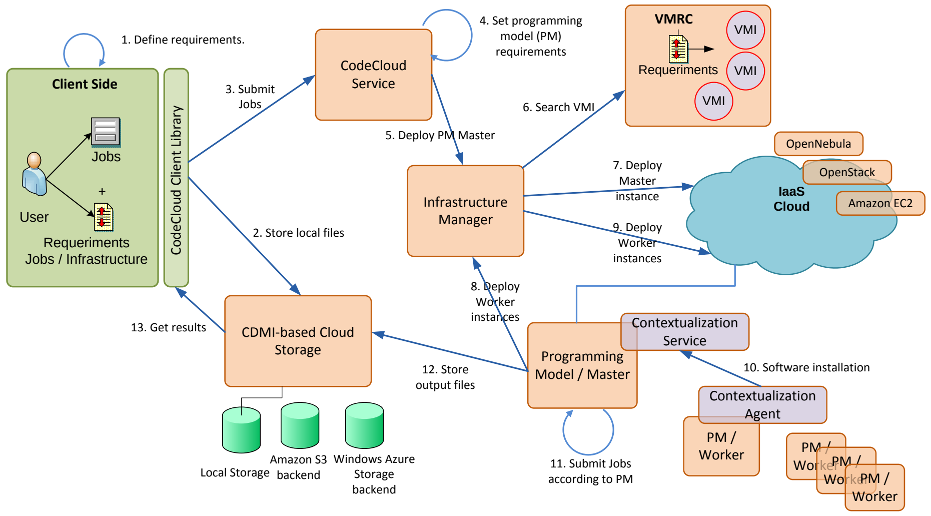

Fig. 1. General Architecture for Programming Model Management on the Cloud. The figure focuses on the master/slave programming model (based on PBS/Torque).

the actual back-end employed (for example Windows Azure Storage or Amazon's Simple Storage Service (S3)). This way, the input files for the jobs will be retrieved from the Cloud storage prior to executing the jobs at destination.

The jobs, that include the description of the desired virtual infrastructure are submitted to the CodeCloud Service (CCS) (step 3). This service is in charge of mediating between the users and a Cloud and orchestrates the rest of the components of the platform. The CodeCloud Service analyzes the programming model specified by the user for the execution of the jobs and it determines the set of additional requirements for the master VM of the underlying virtual infrastructure to support the execution of the jobs under a specific programming model (step 4). In the figure, the focus is set on the Master/Slave programming model by means of PBS/Torque. Therefore, in this particular case the underlying virtual infrastructure should comply with both the requirements imposed by the user (in terms of capabilities of computational resources, and software installed) and the deployment of both PBS/Torque to support the specific programming model.

To manage the deployment of the specific virtual infrastructure employed to support the execution of the jobs, the CodeCloud Service delegates on the Infrastructure Manager (IM) (step 5). The IM, deeply covered in [1], provides the CCS with a set of functions to enable the effective deployment of a computing infrastructure on a Cloud, as well as operations to modify it on demand. The IM takes as input a RADL document that describes the desired virtual infrastructure (in terms of virtual machine's features) and proper credentials to access a Cloud and it performs the deployment of a virtual infrastructure. It currently supports both public Cloud providers such as Amazon Elastic Compute Cloud (EC2) and private Clouds based on OpenNebula and OpenStack.

For that end, the Infrastructure Manager can contact the Virtual Machine image Repository and Catalog (VMRC) service [2], which is a software that enables users to register Virtual Machine Images (VMIs) together with their metadata (hypervisor, OS, applications installed, etc.). This way, the IM can use the specifications issued in the RADL document in order to query the VMRC for the most appropriate VMIs that satisfy a given set of requirements (step 6). For example, a user might specify that a certain job requires a specific GNU/Linux distribution with a particular version of the Java Development Kit. This query can be translated into the VMRC.

Back to the workflow in Figure 1, the Infrastructure Manager deploys an instance which assumes the master role, in the case of the Master/Slave programming model (step 7). The idea is to deploy a self-managed master that manages the life cycle of the execution tasks. That instance is instrumented with a contextualization service that will be later employed to install the dependencies on both the Master the Worker instances. These Worker instances are again launched by the IM, as requested by the Master instance with the additional set of requirements that these nodes require to properly execute the jobs (steps 8 and 9). The Worker instances have a contextualization agent that contacts the contextualization service in order to automatically deploy the required software (for example, the PBS/Torque client together with the dependencies specified by the user) (step 10).

Once the virtual infrastructure is up and running, it is time for the Master instance to submit the jobs to be executed on the Worker instances and monitor their execution (step 11). Notice that the virtual infrastructure is created ad hoc for the job execution request, and it is not shared by other users. The generated output data is stored on the CDMI-based Cloud Storage so that the results can later be retrieved by the client (steps 12 and 13).

3 SLA-aware Platform

The execution of scientific applications in Cloud infrastructures is a process driven by the user's requirements and expectations, and therefore the assurance of the Quality of Service (QoS) levels becomes a relevant topic. In the context of Cloud Computing, QoS is defined as the measure of the compliance of certain user requirement in the delivery of a Cloud resource. In the scenario of execution of scientific jobs, some QoS requirements may be the total time to execute the jobs or the budget used to deploy the resources. In order to provide guarantees in the delivery of the expected QoS level to users, several approaches have been explored. In [3], the authors propose Service Level Agreements (SLAs) as the vehicle for the definition of QoS guarantees, and the provision and management of resources. An SLA is a formal contract between providers and consumers, which defines the resources, the quality of service, the obligations and the guarantees in the delivery of a specific good.

Using this approach, the execution of a Master/Slave scenario may be modeled in a SLA document together with some elasticity rules. An SLA-aware Cloud platform should be able to deploy the master and slaves nodes, executing the jobs and scaling the virtual infrastructure according to the elasticity rules specified by the user in the job definition. That way increasing the number or capacity of running nodes would reduce the execution time of the application when the number of jobs is high, and decreasing the number or capacity of running nodes would reduce the cost of the infrastructure when the number of jobs is low.

To achieve that goal, we introduce Cloud ComPaaS. Cloud ComPaaS is a software platform for the deployment of an SLA-driven layer on top of existing deployments. This platform has been developed as a testbed tool for diverse SLA-aware Cloud resource management techniques inside our research group, although there are plans for releasing it as an open source development.

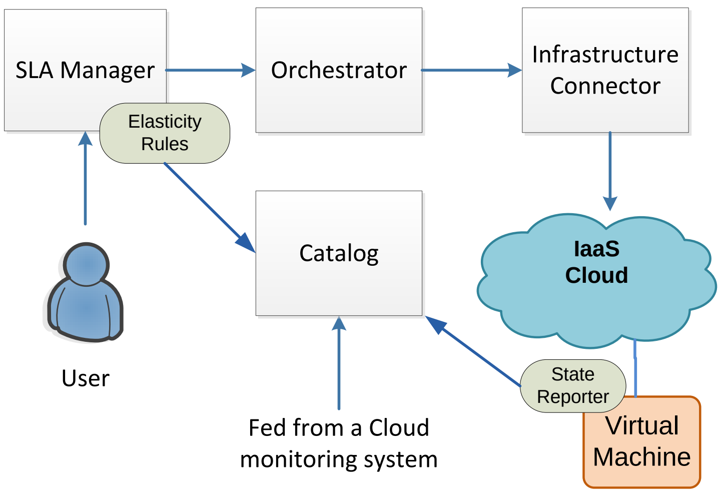

Fig. 2. Architecture for SLA Management.

The architecture of Cloud ComPaaS is designed to produce a set of loosely coupled components that interact among them. The decentralized and distributed nature of its design improves the platform flexibility and resilience, and the loosely coupled interfaces between components improve adaptability and extensibility. Figure 2 introduces the architecture for SLA management. The following sections summarize the main functionality of Cloud ComPaaS by detailing its main components.

3.1 Components

The SLA Manager is the entry point to the Cloud ComPaaS platform. The SLA-driven nature of the platform implies that every interaction between components is performed by means of SLAs (in particular, the WS-Agreement specification is used to describe agreements). The SLA Manager can build agreement documents, check an agreement offer for correctness and register a new SLA. The four basic operations supported by this component are search, create, query and delete.

The Orchestrator is the central component of the platform and acts as a global coordinator with an overall view of the state of the other components.

When a new SLA is accepted by the SLA Manager, a deployment request is sent to the Orchestrator. This component keeps a view of all the available Cloud backends, and performs the scheduling of the resource allocation based on the SLA requirements and the available resources. The Orchestrator manages the deployment process by delegating the allocation operations to an Infrastructure Connector. Hence, this component selects the Cloud backend that will deploy the resources, depending on the user's requirements.

The Orchestrator performs a sequential process for the allocation of resources. This way when a user requests a Virtual Service, the Orchestrator deploys a set of Virtual Machines, retrieve their endpoint references and stores, installs and run the selected software on them.

The Infrastructure Connector is the component in charge of translating the infrastructure SLA (WS-Agreements) into the RADL language in order to properly delegate the deployment of the virtual infrastructure to the aforementioned IM. This way, it is possible to take advantage of the already deployed IM, which enables access to the VMRC catalog, support for multiple IaaS Cloud providers, etc. This connector enables to link ComPaaS with the rest of the system architecture.

3.2 SLA-driven lifecycle assessment in Cloud ComPaaS.

In Cloud ComPaaS the SLAs dictate the lifecycle of Cloud resources. This section describe the lifecycle of resources in a typical usage scenario and details the role fulfiled by each component.

The SLA Manager component provides the entry point and user interface to the operations related to the management of agreements, in the system. Users can search the system for available SLA templates by querying the SLA Manager.

The user can modify the values on this document to generate an agreement offer. The create operation presents the SLA Manager with an agreement offer. The component checks that the offer complies with the agreement template. If this operation fails, the offer is rejected. If the offer is well defined, the SLA Manager sends it to the Orchestrator to schedule its deployment. If this operation fails, for instance because no free resources are available, the offer is rejected.

After an agreement has been accepted and its resources have been allocated, the SLA Manager registers the agreement. Users can hence query the state of agreements that they have sent to the platform (including the rejected ones) and delete an active agreement. Accepted agreements are sent to the Orchestrator, which deploys the resources required by the SLA. If the deployment process completed and the SLA is not rejected during the process, the SLA Manager proceeds to the monitoring of the agreement terms and guarantees states.

The monitoring process consists on the monitoring of service and guarantee terms. Service term monitoring implies gathering monitoring information from the Cloud, which serves to determine the state of each different resource (for instance, the amount of free CPU or memory for a Virtual Machine). The guarantee term monitoring uses the information retrieved from the Cloud to determine the status of the guarantee term (for instance, if the amount of CPU that is free on the Virtual Machine is under a given threshold, then the guarantee is fulfilled). After each monitoring cycle, the SLA Manager performs several additional actions, related with the agreement assessment, such as accounting and billing, or it executes corrective actions, in the case that guarantee terms are violated.

This monitoring continues until one of the following conditions becomes true.

- The SLA is completed. This condition can be met if the SLA is defined for a certain period of time, or when it is defined on the basis of particular objectives (e.g. associated to an individual experiment or execution), which must be completed.
- The SLA is terminated by the consumer (e.g. to avoid consuming resources when the results obtained are sufficient to understand an experiment).
- The SLA is rejected by the provider. Accepted SLA can be rejected by the platform at any time, although this form of termination may involve penalties to the service provider.

These are the three terminal states of an SLA. When the agreement reaches one of these states, the SLA Manager request the undeployment of resources, which is forwarded by the Orchestrator to the corresponding backend. Once the deallocation of resources is performed, the SLA Manager stops the monitoring process and the SLA is eliminated from the system.

4 Elasticity Management

Managing the inherent elasticity that arises from the usage of Cloud infrastructures is of paramount importance in order to accommodate the virtual infrastructure to the dynamic execution requirements of the applications.

Two elasticity modes are currently being considered, as shown in Figure 3. On the one hand, horizontal elasticity enables a set of VMs to dynamically grow/shrink by provisioning/releasing new VMs. On the other hand, vertical elasticity enables a single VM to dynamically increase/decrease its computational capacity (in terms of CPU and memory) depending on the dynamic requirements.

To understand the benefits of horizontal elasticity consider the following scenario. A parameter sweep application (HTC) that dynamically generates jobs to be executed is submitted to the platform. This allocates a certain number of virtual computational resources to allocate the execution of jobs. If the job submission

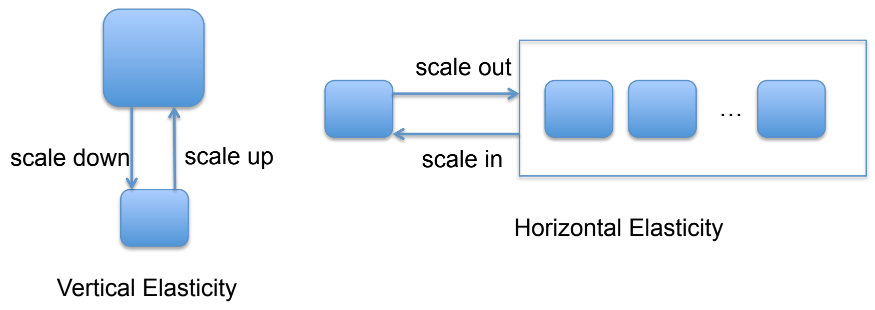

Fig. 3. Elasticity modes considered. Vertical elasticity dynamically increases/reduces the capabilities of a VM. Horizontal elasticity dynamically increases/decreases the number of VMs.

rate starts increasing and the related SLA provides room for increasing the underlying virtual infrastructure, then the virtual platform scales out by provisioning new virtual machines to accommodate the execution of a large number of simultaneous jobs. Whenever the job rate is reduced, so is the virtual infrastructure by scaling in accordingly reducing the allocated VMs.

Concerning vertical elasticity, the scenario proceeds as follows. If the scientific application running on the virtual machine requests a larger amount of memory than initially expected (either because the user underprovisioned the VM or the application memory consumption depends on the evolution of a simulation), then the VM is requested to scale up. This operation increases the amount of memory of the VM without interrupting the execution of the application. Whenever the application frees some memory, the actual allocation of RAM to the VM can also be reduced. This approach would enable to better fit the VM memory allocation to what the application dynamically consumes. In an scenario in which a single physical machine shares the execution of different VMs, this scenario represents a fair approach for resource consumption. Notice that vertical elasticity might involve the migration of the VM to another more powerful computational resource in which the scale up can be performed. This live migration would enable to swap the underlying physical infrastructure prior to scaling up without ever stopping the application.

Vertical elasticity must be supported both by the underlying hypervisors and the OS of the running VM. In the case of the KVM hypervisor employed to run GNU/Linux-based VMs, which is our scenario, memory ballooning is fully supported since kernel 2.6.27. This enables to dynamically change the allocated RAM to a running VM without any noticeably downtime of the VM. In the case of CPU hot plugging, which enables to dynamically add new virtual CPUs (vCPUs) to a running VM, according to KVM documentation there is currently limited support to it.

Horizontal scalability can be managed from the SLA manager, in order to expand a group of VMs in case of reaching some threshold metric conditions. In the case of vertical scalability, the elasticity limits can be defined within an SLA that specifies that VMs should be deployed into an instrumented host. The idea is to deploy an agent running on the physical machine with access to libvirt/virsh and monitoring information about the VMs. Monitoring information can be gathered from the VMs with already existing monitoring software such as Ganglia. Therefore, the agent in the physical machine becomes responsible to manage the vertical elasticity of the VMs running on them, according to the predefined capabilities limits specified for each VM and the capacities of the physical machine itself.

5 Related Work

Currently there are some projects working on creating Platform as a Service (PaaS) environments to enable the users to access Cloud technologies. There are some well-known commercial solutions such as Google App Engine (GAE) or Microsoft Azure that provides PaaS solutions over their own commercial Cloud infrastructures. Other commercial alternative is Aneka [4] provided by Manjrasoft, which is a software platform for deploying Clouds and developing applications on top of it. It provides a runtime environment and a set of APIs that allow developers to build .NET applications that leverage their computation on either public or private clouds.

Nimbus [5] is an open source IaaS system that allows a client to lease remote resources by deploying VMs onto those resources. It also enables to automate the configuration and deployment of different software packages using the Nimbus Context Broker, enabling to deploy Clusters of PCs. These kind of solutions enable to create and contextualize virtual infrastructures but does not support any type of programming model.

Simple API for Grid Applications (SAGA) [6] is a programming system that provides a high level API for users to use C++, Python, or Java languages to interact with distributed computing resources. It was designed to be used in grid environments but recently this project has added support for cloud infrastructure interaction. A MapReduce implementation using SAGA has been created, enabling users to launch MapReduce applications in all the environments supported by SAGA.

Sector and Sphere [7] is a cloud framework specifically designed for writing applications able to utilize the stream processing paradigm (similar approach to MapReduce). Sector is a distributed file system that manages data across physical compute nodes at the file level, and provides the infrastructure to manipulate data. Sphere, on the other hand, provides the framework to utilize the stream processing paradigm for processing the data residing on Sector. The Sphere system is composed of Sphere Processing Engines (SPEs) running on the same physical nodes as the Sector file system. These two projects provide support for programming models, but only one type is supported (MapReduce and stream processing), moreover it does not enable to personalize the VMs using user requirements. AppScale is an open source implementation of the Google App Engine (GAE) PaaS cloud technology. As a new development over AppScale, Neptune is a domain specific language that automates configuration and deployment of existing HPC software via cloud computing platforms. It is an extension of the Ruby programming language that enables to launch scientific applications with the MPI and MapReduce programming models. In this case two programming models are supported, and also enable to extend the language, but similar to the previous projects it does not enable to personalize the VMs with user requirements.

In our proposed architecture, SLAs are employed to deliver QoS. Concerning the usage of SLAs, the study of QoS in Cloud Computing focuses on providing Cloud platforms and deployments with mechanisms to the enforcement of the requirements of Cloud users. These studies have been made on two major models of Cloud delivery, Cloud services and Cloud adapted workflows. Even though Cloud services are the focus of the offer of the major Cloud providers, the QoS assessment in Cloud workflows represents a more complex scenario that includes stricter requirements. Therefore the techniques and algorithms developed for Cloud workflows can be readily adopted for the Cloud services scenario.

An algorithm to select services that comply with user requirements from a pool is presented in [8]. The paper proposes modeling the QoS parameters of services and enable users to query for services in different service pools. The SPSE algorithm implements several stages to enable users to select Cloud services according to QoS constraints. The system aims not only to provide users with a mechanism to retrieve services that meet their QoS criteria, but to serve users with the services that best fit their preferences, therefore enhancing the user experience.

In [9], the authors perform a deep analysis of the QoS assessment in the delivery of Cloud services, and focuses on very specific problems of the Cloud platforms. They propose Service Level Agreements as the vehicle for the QoS level specification. This formal document captures the requirements of an actor respect to another, and provides a common framework for the definition of QoS criteria, obligations and penalties. They also model the transitive relationship between Cloud providers and Cloud users, and Cloud users as application providers and end-users as application consumers. Even though the existence of this relationship has been cited in [10], the paper analyses its impact in the QoS assessment.

Finally, in [11] the authors present the architecture and working prototype of the WfMS platform for the QoS-aware execution of workflows. The platform captures the QoS criteria for the execution of workflows in SLA documents, and a monitoring system captures information related to these criteria from the running components and publishes it to an index for other components to retrieve.

6 Conclusions and Future Work

This work has introduced a proposal of architecture to abstract the details of scientific application execution on Cloud infrastructures. The architecture aims at simplifying the execution of applications under different programming models, but this paper has focused on the Master/Worker HTC model. The platform features services ranging from (i) the description of jobs and infrastructures from high level infrastructure-agnostic declarative languages, (ii) SLA-management in order to satisfy the requirements imposed by the user, (iii) infrastructure management to deploy VMs regardless of the underlying IaaS Cloud provider, (iv) cataloging of

VMIs in order to search for the most appropriate VMIs that satisfy a given set of requirements and (v) support to horizontal elasticity and partial support to vertical elasticity based on SLAs, in order to better fit the underlying infrastructure to the dynamic requirements of the running applications.

The implementation of the platform is an ongoing work. Some components have been completely implemented and even released as open source components, such as the VMRC system[1] to catalog VMIs. The Infrastructure Manager together with the RADL language has also been completely developed and described in [1]. Concerning the Cloud ComPaaS, a prototype implementation based on WS-Agreements with horizontal elasticity support is already available [12]. Vertical elasticity is currently an ongoing work and initial progress has been made.

Future works involve extending the proposed platform to implement support to other programming models. This would certainly benefit scientific applications that involve other execution patterns such as Workflows or MapReduce.

Acknowledgements

The authors wish to thank the financial support received from the Ministerio de Economía y Competitividad for the project CodeCloud (TIN2010-17804) and to the Generalitat Valenciana for the project GV/2012/076. This work has been developed under the support of the program Formación de Personal Investigador de Carácter Predoctoral grant number BFPI/2009/103, from the Conselleria d'Educació of the Generalitat Valenciana.

References

1. de Alfonso, C., Caballer, M., Alvarruiz, F., Moltó, G., Hernández, V.: Infrastructure Deployment Over the Cloud. In: 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2011). (2011) 517–521
2. 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)
3. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems **25**(6) (2009) 599–616
4. Vecchiola, C., Chu, X., Buyya, R.: Aneka: A Software Platform for .NET-based Cloud Computing. (2009) 30
5. 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)
6. Kaiser, H., Merzky, A., Hirmer, S., Allen, G., Seidel, E.: Poster reception—The SAGA C++ reference implementation. In: Proceedings of the 2006 ACM/IEEE conference on Supercomputing - SC '06, New York, New York, USA, ACM Press (2006) 184

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

7. Gu, Y., Grossman, R.: Sector and Sphere: The Design and Implementation of a High Performance Data Cloud. Philosophical transactions. Series A, Mathematical, physical, and engineering sciences **367** (2009) 2429–2445
8. Zhao, L., Ren, Y., Li, M., Sakurai, K.: Flexible service selection with user-specific QoS support in service-oriented architecture. Journal of Network and Computer Applications **35**(3) (2012) 962–973
9. Wu, L., Kumar Garg, S., Buyya, R.: SLA-based admission control for a Software-as-a-Service provider in Cloud computing environments. Journal of Computer and System Sciences **78**(5) (2012) 1280–1299
10. 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)
11. Gogouvitis, S., Konstanteli, K., Waldschmidt, S., Kousiouris, G., Katsaros, G., Menychtas, A., Kyriazis, D., Varvarigou, T.: Workflow management for soft real-time interactive applications in virtualized environments. Future Generation Computer Systems **28**(1) (2012) 193–209
12. García, A., de Alfonso, C., Hernández, V.: Design of a Platform of Virtual Service Containers for Service Oriented Cloud Computing. In: Cracow Grid Workshop '09 Proceedings. (2010) 20—-27