Dynamic Deployment of Custom Execution Environments in Grids *

Ruben S. Montero, Eduardo Huedo and Ignacio M. Llorente
Facultad de Informática
Universidad Complutense de Madrid
28040 Madrid, Spain.
rubensm@dacya.ucm.es, ehuedo@fdi.ucm.es, llorente@dacya.ucm.es

Abstract

One of the most important obstacles when porting an application to the Grid is its highly heterogeneous nature. This heterogeneity usually means an increase of the cost of both the application porting cycle and the operational cost of the infrastructure. Moreover, the effective number of resources available to a user are also limited by this heterogeneity. In this paper we presents two approaches to tackle these problems: (i) an straightforward deployment of custom virtual machines to support the application execution; (ii) and a new architecture to provision computing elements that allows to dynamically adapt them to changing VO demands. Experimental results for both approaches on prototyped testbed are discussed. In particular, the on-demand provision of computing elements show less than a 11% overall performance loss including the hypervisor overhead.

1. Introduction

In the last decade, the distributed computing scene has been dominated by large size Grid infrastructures deployed within the context of international research projects, such as Enabling Grids for E-sciencE (EGEE), TeraGrid or Open Science Grid. These projects have achieved unseen levels of resource sharing, and potentially provide an unprecedented amount of processing and storage resources to applications.

However, scientific applications have obtained different levels of success when using the computational resources provided by the Grid infrastructure. Probably, the growing heterogeneity of the organizations that join a Grid is the most important problem that arises when porting an application. Grid resources do not only differ in their hardware but also in their software configurations (operating system, libraries, and applications). The result is an infrastructure which is difficult to maintain and program.

Given the different requirements of different Virtual Organizations (VO), this heterogeneity usually means:

- An increase of the cost and length of the application development or porting cycle. New applications have to be tested in a great variety of environments where the developers have limited configuration capabilities.
- A limitation of the effective number of resources available to a user. Usually different VOs require different software configurations. Only those sites supporting a given VO are willing to invest the effort needed to install, configure and maintain custom software configurations.
- An increase of the operational cost of the infrastructure. New sites joining a Grid infrastructure not only need to install, configure and upgrade the basic middleware but also software tools related to the VOs that the site will support.

The difficulties outlined above had been previously identified and are currently being tackle by several projects. The different approaches can be classified in three main categories, namely:

- *Software-environment configuration systems*. The most important project in this category is SoftEnv. SoftEnv is a system that makes it easier for users to define what applications they want to use, and easier for administrators to make applications available to users. Although, systems like SoftEnv help applications to use different software environments, they do not completely solve any of the previous listed problems.
- *Deployment of software-environment overlays*. This solution consists in deploying custom software con-

*This research was supported by Consejería de Educación de la Comunidad de Madrid, Fondo Europeo de Desarrollo Regional (FEDER) and Fondo Social Europeo (FSE), through BIOGRIDNET Research Program S-0505/TIC/000101, by Ministerio de Educación y Ciencia, and through the research grant TIN2006-02806, and by the European Union through the research project RESERVOIR Contract Number 215605

figurations in the user-space. In this case the deployment is managed by a general-purpose resource management system. Probably the most important examples in this category are Condor Glidein [8] (to deploy Condor pools). This approach is limited by the fact that the software must be installed in the user-space. In addition, compatibility issues are shifted from the application layer to the overlaid software layer.

- *Virtual Machine technologies.* Virtual Machines (VM) add a new abstraction layer that allows partitioning and isolating the physical hardware resources. The integration of virtual machines in Grid environments has been previously explored by several works. For example, the In-VIGO project [6] establishes a basic layer of virtual Grid resources upon which any grid middleware can be deployed. The Virtual Workspace Service [10], exposes the functionality needed to manage workspaces, an abstraction of execution environments implemented through VMs (see Section 4 for other related projects in this context).

In this work we review two different solutions to provide scientific applications with custom execution environments in Grids. These approaches preserves the current middleware-stack of common Grid deployments, which eases their adoption and allows the coexistence of different application paradigms.

First we present in Section 2, an straightforward deployment of virtual machines in a Globus Grid. Then in Section 3, we describe a new architecture for the dynamic provisioning of computational services on a Grid infrastructure. The system leverages virtualization technologies to provide flexible support for different VOs. In Section 4 we discuss some related work, and the paper ends with some conclusions in Section 5.

2. Straight Forward Deployment of Virtual Machines

In this section we describe the deployment and execution management of single virtual machines in a Grid infrastructure. This approach consists in encapsulating a virtual machine within a grid job. The target application will then be executed in the deployed VM. In this way, VMs can ensure the correct execution of the application by providing application-specific software configurations in a *well-known* environment.

In this case, the life-cycle and control of the underlying VM is managed by a general purpose job meta-scheduler. The system incorporates the functionality and benefits offered by a general job management framework. So, the genuine characteristics of a Grid infrastructure (i.e.dynamism, high fault rate, heterogeneity) are naturally considered in the proposed solution.

This strategy does not require additional middleware (apart from the virtualization layer) to be deployed, as it is based on well-tested procedures and standard services. Moreover, it is not tied to a given virtualization technology. However managing a VM as a grid job presents some drawbacks:

- The underlying local resource management system is not aware of the nature of the job itself. Therefore, some of the potential benefits offered by the virtualization technology (e.g. server consolidation) are not fully exploited.

- This approach is only suitable for single process batch jobs, which naturally arise in High Throughput Computing problems, or workflow computations. However this use case represents a small fraction of the applications that can potentially benefit from the virtualization technology in a Grid.

2.1. Architecture of the System

We will use the GridWay meta-scheduling system [9] to control and manage the deployment of virtual machines. GridWay allows unattended, reliable, and efficient execution of jobs on heterogeneous and dynamic Grids. It performs transparently to the end user all the job scheduling and submission steps [12], namely: resource discovery and selection, and job preparation, submission, monitoring, migration and termination.

The VM that contains the application execution environment is deployed as a grid job using a classical three step schema: *prolog*, for creating the remote experiment directory and transferring the executable and input files; *wrapper* for executing the actual job and obtaining its exit code; and *epilog* for transferring back output files and cleaning tasks.

GridWay schedules a grid job (or VM in our case) as follows: it holds a list with the available resources in the Grid and their characteristics. This list is periodically updated by querying the information services to monitor and discover grid hosts. GridWay filters out those resources that do not have free slots, or do not meet the job (VM) requirements. Then it sorts the remaining hosts according to an user-supplied rank expression. The highest ranked resource is used to dispatch the job.

In order to efficiently schedule a VM, the description of the grid job must include all the requirements of the VM within. So, the job template used by GridWay must specify the Hypervisor type (e.g. Xen [1] or KVM [2]) and version, and other hardware requirements (e.g. free memory or CPU load). Additionally, the Grid Information Systems must be modified to provide such information.

In the following analysis we assume that application-specific environments are prepared following an install once deploy many approach. The VMs are distributed at VO level and are available at the remote resources. In general, the images could be downloaded, as any other input file, in the *prolog* phase from the client or from an scientific virtual appliance repository.

The execution of the application can be summarized as follows (see Figure 1):

1. The application is defined using a grid job template that must include the execution environment to run it (specified by the VM image), the set of input and output files, and the requirements to deploy the VM.

2. Gridway selects a grid resource to execute the job based on the specified requirements and preferences. Before actually executing the job on the resource an optional *pre-wrapper* phase can be defined to perform advanced job configuration routines. This phase consists in an user defined program that is executed on the cluster front-end. In our case, this program could check the availability of a given image, and transfers it from a GridFTP repository if needed.

3. The input files of the application are transferred to the selected resource.

4. The *wrapper* program is submitted through Grid execution services (WS-GRAM in our case). The *wrapper* starts (or restores) the specified virtual machine and waits for its activation. Once the VM is up and running it copies all the input files needed by application to the VM and executes the actual job. Finally, the output files are transferred back to the physical cluster file system and it shuts down (or suspends to disk) the virtual machine.

5. The results are copied back to the client host in the *epilog* phase.

2.2. Overhead Analysis

The behavior of the previous deployment strategy will be analyzed on a research testbed based on the Globus Toolkit 4 and GridWay 5. The testbed consists of two resources: a client host, and a SGE cluster for processing purposes. The main characteristics of these machines are described in Table 1. These hosts are connected by a Fast Ethernet campus network.

In order to study the overhead imposed by these approach we will consider the execution of a Scientific Analysis System (SAS) workload, that analyzes XMM-Newton X-ray satellite data [5]. The experiment consists of the executable; the input files with observation data (20MB); and an output file (14MB) with observation events to perform post-processing.

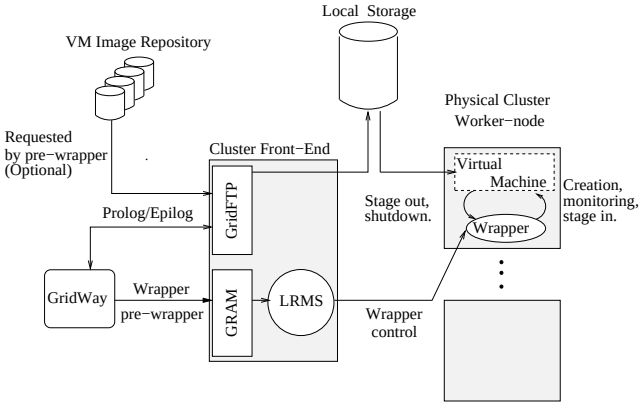

Figure 1. Schematic representation of the execution within a virtual machine in a Grid environment.

Let us first consider the raw virtualization overhead, which includes the time to transfer job files to/from the execution environment. In this case we will consider that the execution environments (VMs) are set-up and running (e.g. pre-deployed by a system as described in Section 3). The overall execution of the application within the VM involves a 20% increment in execution time, compared with the standard execution (150sec.)

Usually, the execution environments will be dynamically created and destroyed to satisfy user requests. In this case, the time to start and stop the VM must be considered. In general, this time will depend on the services started on the VM; for the SAS-execution the VM has been tailored to only included those services needed by the application. The overall start/stop overhead is 80 seconds, and can be further reduced to 60 sec. by caching the VM (suspend/resume).

3. Dynamic Provisioning of Computing Elements

The solution described in Section 2 presents several drawbacks like a limited use of the potential benefits offered by the virtualization technology, or a limited application range. These limitations can be overcome by tackling the problem at the infrastructure level. In this way, Grid site administrators would be able to satisfy the following requirements:

- *Heterogeneous configuration demands*, to support VO-specific worker node configurations (e.g. operating system, libraries or post-processing utilities) and so

Table 1. Summary of characteristics of the testbed resources.

Host	CPU	Memory	Service Configuration
client	PIV HT 3.2GHz	512MB	GT4, GridWay
front-end	PIV HT 3.2GHz	512MB	GT4, SGE, NIS, NFS, DHCP
worker-node	PIV HT 3.2GHz	2GB	Xen3.0 testing

provide on-the-fly custom application execution environments.

- *Performance partitioning*, to isolate and partition the performance of the physical resources they devote to different Grid infrastructures or VOs. Moreover, the infrastructure should be able to balance the amount of resources allocated to each VO in terms of their dynamic requests.

To achieve these two goals, we propose a multi-layer architecture to dynamic provision virtual worker nodes to clusters inside a Grid. The infrastructure, given a set of VO execution environments, is capable of deciding the number and the kind of worker nodes to be created on each cluster. This way, the computing elements of a grid can be adapted to fit the changing software requirements and computational demands.

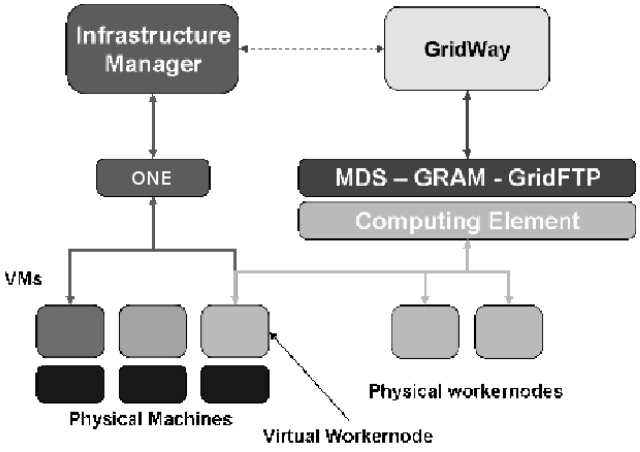

Figure 2. Schematic representation of the deployment of a virtual worker node in a Grid.

3.1. Architecture of the System

As in the previous case, at the top layer a meta-scheduler is responsible for the management, control and monitoring of the scientific application. In the prototype implementation of the architecture we will use GridWay, as it features the scheduling and fault detection capabilities required by the system. The meta-scheduler interacts with the Grid resources using basic Grid middleware services, those provided by Globus Toolkit 4.

The underlying Grid resources are configured with different queues for each VO (this is a common setup in current Grids). So, new virtual worker-nodes can be easily deployed using the corresponding VO appliance. Note that in this way, several software configurations can coexist on a single cluster and a job can be executed in the correct one by just specifying the queue name.

When a new worker-node is deployed it registers in the computing element, and this information is pushed to the Information Service (the MDS service in our case). Then GridWay using its monitoring capabilities is able to detect the new slot and submit applications to the corresponding queue through GRAM. This process is depicted in Figure 2

Finally, the Infrastructure Manager module completes the system architecture. This component is responsible for adapting the Grid computational services to the dynamic Grid computing demands. The Infrastructure Manager decides when to add new worker nodes (and their type) to a given computing element (cluster queue). The deployment and management of the VMs is performed through the OpenNebula virtual infrastructure engine [3]. The ability of OpenNebula to deploy several copies of a single image along a physical cluster, with a pre-set pool of host names and IPs, keeps integration with LRMS layer simple.

The following list details the actions that takes place when the Information Manager decides to add a new worker node to the computing element:

1. The Infrastructure Manager request a new VM to OpenNebula (using a predefined appliance for the VO). OpenNebula determines the best node to run the virtual machine, based on the resources requested (e.g memory).

2. If the VM image (appliance) is not local to the host system, it accesses the image server via a suitable protocol (e.g. GridFTP) and obtains a copy.

3. Once the image has been transferred, the physical node's DHCP server configuration file is altered in order to establish VM's IP and hostname. When these operations conclude, the VM is booted.

4. When the VM has been deployed and it is running, it registers on the LRMS frontend as an execution host.

5. After a given time, the Grid information system (MDS) detects the new node and publishes it.

6. Finally, the meta-scheduler (GridWay) will refresh the information of the available Grid resources, and detect the new worker node. Then, according to the scheduling policies, it will allocate jobs on this new resource by interfacing with the Grid execution service (GRAM).

3.2. Overhead Analysis

The aim of the experiments presented in this section is to obtain a clear understanding of the interaction of all the components that form the architecture. The testbed used to perform the experiments is the same as the one described in Section 2. The experiment consists in analyzing the process of the deployment of a VO-specific SGE workernode, and its eventual use at the Grid level.

Let us start by measuring the local deployment time of a VM. With the above configuration, we have obtained an average total deployment time of a virtual worker node of 36 seconds. This time includes the OpenNebula overhead (15 seconds including the image propagation time), the VM booting time (19 seconds including the network setup), and the time to register to the SGE front-end (2 seconds).

The time to register the new slot in the Grid Information system (MDS4) is about 170 seconds. It is worth pointing out that MDS publishing time is greater than the time employed on deploying one VM plus SGE register time. Therefore, when sequentially deploying several VMs both times overlap, producing an additional time saving. The MDS and GridWay overhead can be limited by adjusting their refresh polling intervals. Therefore, the total time to deploy a new worker-node its about 300 seconds. Note that this overhead, unless the previous solution, can be distributed among different applications as the VM can be reused.

When switching down a worker node, the same steps are accomplished. In this case, the time until the operation is accomplished at the machine layer is greatly reduced, from 36 to 7 seconds. However, time until LRMS detects the lack of the VM is incremented, from 2 to about 150 seconds. It is interesting to note that the meta-scheduler could assign jobs to the cluster during the worker node shutting down time. In this case the meta-scheduler should be able to re-schedule this job to another resource. Finally, the performance penalty imposed by Virtualization technology is the same of that analyzed previously.

4. Related Work

The are several works that analyzes the dynamic provision of a computational cluster, to adapt its size to the workload. Jeffrey Chase et al., from Duke University, describe in [7] a cluster management software called COD (Cluster On Demand). In our case, the solution is based on virtual machines and the system is studied from the application perspective.

The Edge Services [11] provides a framework to deploy VO-dedicated servers executed at the network edge (public to private boundary) of each site. These VO servers would run VO specific software packages with custom configurations. The edge services are dynamically deployed using the Virtual Workspace Service. In this paper, our goal is not to create a new service (Edge Service) but to improve and adapt existing ones, supporting different VOs in a shared physical infrastructures.

Finally, Amazon Elastic Computing Cloud [4] provides a remote VM execution environment. It allows to execute one or more “Amazon Machine Images” on their systems, providing a simple web service interface to manage them. It would be possible to employ a grid-enabled Amazon Machine Image, and create as many instances as needed, getting on-demand resources in case the physical hardware cannot satisfy a peak demand.

5. Conclusions

In this work we have presented two different approaches to deploy custom execution environments in Grids. The first approach provides an straight forward deployment of virtual machines tailored for the execution of a given application. Although this alternative can be easily deployed and saves administration efforts it presents several drawbacks like a limited use of the potential benefits offered by the virtualization technology.

We have presented a new Grid architecture for the dynamic provisioning of computing elements. The benefits of this architecture is a flexible Grid able of supporting different VOs on a shared and configurable infrastructure, while reducing its operational costs.

The results obtained on the performance tests show that the proposed architecture and technologies represent a feasible solution wich induces a limited overhead. At the infrastructure level this new architecture provides the following benefits: cluster consolidation because multiple virtual working nodes can run on a single physical resource; cluster partitioning because the physical resources of a cluster could be used to execute virtual working nodes bound to different virtual clusters; and support for heterogeneous workloads.

References

[1] http://www.xen.org.

[2] http://kvm.qumranet.com/kvmwiki.

[3] http://www.opennebula.org.

[4] www.amazon.com/ec2.

[5] XMM-Newton Science Analysis Software. European Space Astronomy Center. Available at http://xmm.vilspa.esa.es/sas/.

[6] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu. From Virtualized Resources to Virtual Computing Grids: The In-VIGO system. *Future Generation Computer Systems*, 21(6), April 2005.

[7] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle. Dynamic virtual clusters in a grid site manager. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, page 90, Washington, DC, USA, 2003. IEEE Computer Society.

[8] J. Frey et al. Condor/G: A Computation Management Agent for Multi-Institutional Grids. In *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing (HPDC10)*, August 2001.

[9] E. Huedo, R. S. Montero, and I. M. Llorente. A Framework for Adaptive Execution on Grids. *Intl. J. Software – Practice and Experience (SPE)*, 34(7):631–651, 2004.

[10] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid. *Scientific Programming Journal*, 13(4):265–276, 2005.

[11] K. Keahey, T. Freeman, A. Rana, M. Norman, F. Wrthwein, and R. Gardner. Edge services framework for osg. *OSG Document 167-v1*, June 2005.

[12] J. M. Schopf. Ten Actions when Superscheduling. Technical Report GFD-I.4, Scheduling Working Group – The Global Grid Forum, 2001.