# Building a Virtualized Distributed Computing Infrastructure by Harnessing Grid and Cloud Technologies

Alexandre di Costanzo, Marcos Dias de Assunção, and Rajkumar Buyya
Grid Computing and Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia

## *Abstract*

In this article, we present the realization of a system, termed as InterGrid, for interconnecting distributed computing infrastructures by harnessing virtual machines. The InterGrid aims to provide an execution environment for running applications on top of the interconnected infrastructures. The system uses virtual machines as the building blocks to construct execution environments that span multiple computing sites. An execution environment is a network of virtual machines created to fulfill the requirements of an application, thus running isolated from other execution environments. These environments can be extended to operate on Cloud infrastructure, such as Amazon EC2. The article provides an abstract view of the proposed architecture and its implementation; experiments show the scalability of an infrastructure managed by InterGrid and how the system can benefit from using Cloud infrastructure.

## *Introduction*

Over the last decade, the distributed computing area has been characterized by the deployment of large-scale Grids such as EGEE [1] and Grid'5000 [2],. Such Grids have provided the research community with an unprecedented number of resources, which have been used for various scientific endeavors. Several efforts have been made to enable interoperation between Grids by providing standard components and adapters for secure job submissions, data transfers, and information queries [3]. Despite these efforts, the heterogeneity of hardware and software has contributed to increasing complexity of deploying applications on these infrastructures. Moreover, recent advances in virtualization technologies [4] have led to the emergence of commercial infrastructure providers, also known as Cloud computing [5]. Handling the ever growing demands of

distributed applications while addressing heterogeneity, remains a challenging task that can require resources from both Grids and clouds.

In previous work, we presented an architecture for resource sharing between Grids [6] inspired by the peering agreements established between Internet Service Providers (ISPs) in the Internet, through which ISPs agree to allow traffic into one another's networks. This work presents the realization of the architecture, which is termed as InterGrid and relies on InterGrid Gateways (IGGs) that mediate access to resources of participating Grids. The InterGrid also aims at tackling the heterogeneity of hardware and software within Grids. The use of virtualization technology can ease the deployment of applications spanning multiple Grids as it allows for resource control in a contained manner. In this way, resources allocated by one Grid to another are used to deploy virtual machines. Virtual machines also allow the use of resources from Cloud computing providers.

We first introduce the concepts on virtualization and Cloud computing. Then, we describe the InterGrid project (www.gridbus.org/intergrid), which aims to provide an infrastructure for deploying applications on several computing sites, or Grids, by using virtualization technologies. These applications run on networks of virtual machines or execution environments created on top of the physical infrastructure. Next, the system design and implementation are presented. To illustrate the different interactions among the system components, we describe the InterGrid at runtime. Finally, experimental results demonstrate the system scalability and show how applications can combine resources from two cloud provider sites.

## *Background and Context*

### Virtualization Technology and Infrastructure as a Service

The increasing ubiquity of Virtual Machine (VM) technologies has enabled the creation of customized environments atop physical infrastructure and the

emergence of business models such as Cloud computing. The use of VMs brings several benefits such as:

(i) Server consolidation, allowing workloads of several under-utilized servers to be placed in fewer machines;

(ii) The ability to create VMs to run legacy code without interfering with other applications' APIs;

(iii) Improved security through the creation of sandboxes for running applications with questionable reliability; and

(iv) Performance isolation, thus allowing a provider to offer some guarantees and better QoS to customers' applications.

Existing virtual-machine based resource management systems can manage a cluster of computers within a site allowing the creation of virtual workspaces [7] or virtual clusters [8]. They can bind resources to virtual clusters or workspaces according to a user's demand. These systems commonly provide an interface through which one can allocate VMs and configure them with the operating system and software of choice. These resource managers, also named Virtual Infrastructure Engines (VIE), allow the user to create customized virtual clusters using shares of the physical machines available at the site.

As explained earlier, virtualization technology minimizes some security concerns inherent to the sharing of resources among multiple computing sites. We utilize virtualization software to realize the InterGrid architecture because existing cluster resource managers relying on VMs can provide us with the building blocks, such as the availability information, required for the creation of virtual execution environments. In addition, relying on VMs eases the deployment of execution environments on multiple computing sites; the user application can have better control over the software installed on the resources allocated without compromising the operation of the hosts' operating systems at the computing sites.

Virtualization technologies have also facilitated the realization of Cloud Computing services. Cloud computing includes three kinds of services accessible

by the Internet: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). This work considers only IaaS. IaaS aims to give provide computing resources or storage as a service to users. One of the major players in Cloud computing is Amazon with its Elastic Compute Cloud (EC2)[1], which comprises several data centers located around the world. EC2 allows users to deploy VMs on demand on Amazon's infrastructure and pay only for the computing, storage and network resources they use.

**Related Work**

Existing work has shown how to enable virtual clusters that span multiple physical computer clusters. A broker is responsible for managing a virtual domain (*i.e.* a virtual cluster) in VioCluster [1], and can borrow resources from another broker. Brokers have borrowing and lending policies that define when machines are requested from other brokers and when they are returned, respectively.

Systems for virtualizing a physical infrastructure are also available. Montero *et al.* [2] investigated the deployment of custom execution environments using Open Nebula. They investigated the overhead of two distinct models for starting virtual machines and adding them to an execution environment. Montero *et al.* [3] also used GridWay to deploy virtual machines on a Globus Grid; jobs are encapsulated as virtual machines. Montero *et al.* showed that the overhead of starting a virtual machine is small for the application evaluated.

Several load-sharing mechanisms have been investigated in the distributed systems realm. Iosup *et al.* [4] proposed a matchmaking mechanism for enabling resource sharing across computational Grids. Surana *et al.* [5] addressed the load balancing in DHT-based P2P networks.

[1] P. Ruth, P. McGachey, and D. Xu. VioCluster: Virtualization for dynamic computational domain. In *IEEE International on Cluster Computing (Cluster 2005)*, pages 1–10, Burlington, USA, September 2005. IEEE.

[2] R. S. Montero, E. Huedo, and I. M. Llorente. Dynamic deployment of custom execution environments in Grids. In *2nd International Conference on Advanced Engineering Computing*

[1] http://aws.amazon.com/ec2/

*and Applications in Sciences (ADVCOMP '08)*, pages 33–38, Valencia, Spain, September/October 2008. IEEE Computer Society.

[3] A. Rubio-Montero, E. Huedo, R. Montero, and I. Llorente. Management of virtual machines on globus Grids using GridWay. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007)*, pages 1–7, Long Beach, USA, March 2007. IEEE Computer Society.

[4] A. Iosup, D. H. J. Epema, T. Tannenbaum, M. Farrellee, and M. Livny. Inter-operating Grids through delegated matchmaking. In *2007 ACM/IEEE Conference on Supercomputing (SC 2007)*, pages 1–12, New York, USA, November 2007. ACM Press.

[5] S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, and I. Stoica. Load balancing in dynamic structured peer-to-peer systems. *Performance Evaluation*, 63(3):217–240, 2006.

## InterGrid Concepts

As this work presents the realization of the InterGrid, this section provides an overview of the proposed architecture; details about the architecture are available in previous work [6].

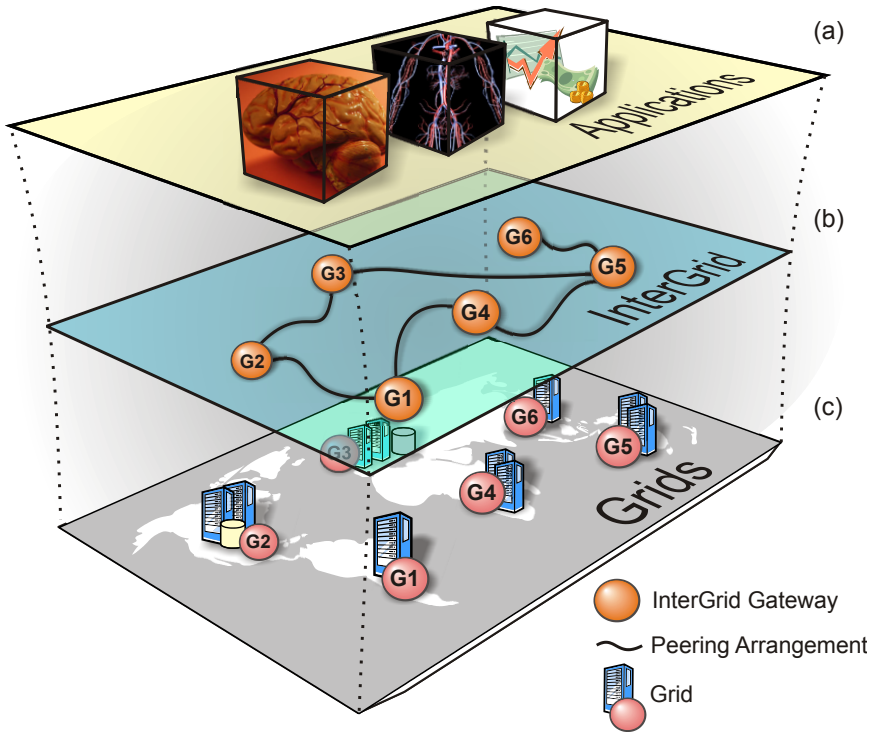


**Figure 1: Abstract view of the software layers of the InterGrid.**

Figure 1 depicts the scenario considered by the InterGrid. The InterGrid aims to provide a software system that allows the creation of execution environments for various applications (a) on top of the physical infrastructure provided by the participating Grids (c). The allocation of resources from multiple Grids to fulfill

the requirements of the execution environments is enabled by peering arrangements established between gateways (b).

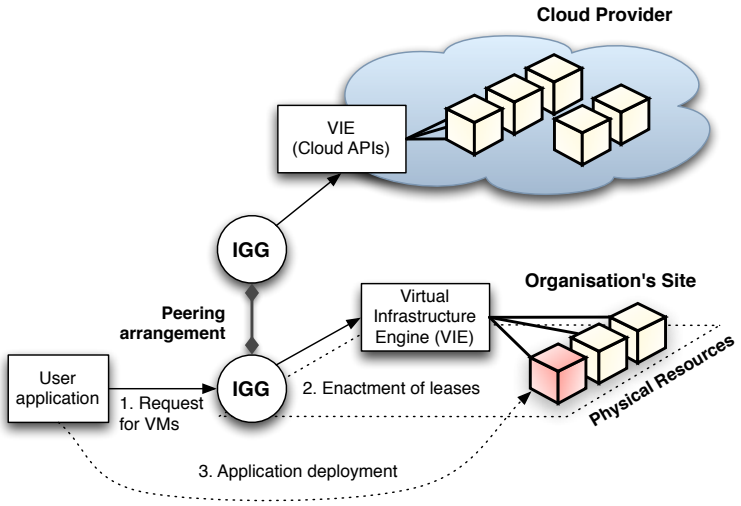


**Figure 2: Application deployment**

A Grid has pre-defined peering arrangements with other Grids, managed by IGGs and, through which they co-ordinate the use of resources of the InterGrid. An IGG is aware of the terms of the peering with other Grids; selects suitable Grids able to provide the required resources; and replies to requests from other IGGs. Request redirection policies determine which peering Grid is selected to process a request and a price at which the processing is performed. An IGG is also able to allocate resources from a Cloud provider. Figure 2 illustrates a scenario where an IGG allocates resources from an organization's local cluster for deploying applications. Under peak demands, this IGG interacts with another that can allocate resources from a cloud computing provider.

Although applications can have resource management mechanisms of their own, we consider a case where the resources allocated by an application are used for the creation of a Distributed Virtual Environment (DVE), which is a network of virtual machines that runs isolated from other DVEs. Therefore, the allocation and management of the acquired resources is performed on behalf of the application by a component termed DVE Manager.

### *InterGrid Realization*

The InterGrid gateway has been implemented in Java and its main components are depicted in Figure 3.

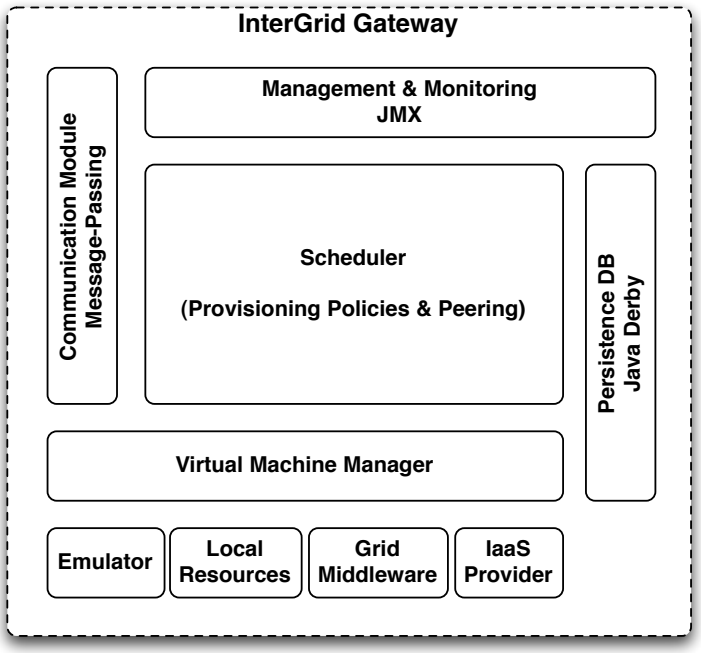


**Figure 3: Main components of the InterGrid Gateway.**

**Communication Module:** is responsible for message-passing. This module receives messages from other entities and delivers them to the components registered as listeners for those types of messages. It also allows components to communicate with other entities in the system by providing the functionality to send messages. Message-passing helps making gateways loosely coupled and building more failure-tolerant communication protocols. In addition, sender and receiver are de-coupled, making the whole system more resilient to traffic bursts. All the functionalities of the module are handled by one central component, the *Post Office.* When the Post Office receives an incoming message, the message is associated to a thread and then forwarded to all listeners. Threads are provided by a pool and if the pool is empty, messages are put in a queue to wait for available threads. Listeners are message handlers, which deal with messages. All listeners are notified when a new message arrives. Listeners can individually decide to process the message. As messages are asynchronous

and sent/served in parallel, there is no order or delivery guaranties. It is the responsibility of communicating components to handle those properties. For instance, the scheduler uses unique message identifiers to manage request negotiations.

**Management and Monitoring:** are performed via Java Management Extensions (JMX). JMX is a standard API for management and monitoring of resources such as Java applications. It also includes remote secure access, so a remote program can interact with a running application for management purposes. The gateway exports, via JMX, management operations such as configuring peering, connecting/disconnecting to another gateway, shutdown the gateway, and managing the virtual machine manager. All these operations are accessible via JConsole, which is a graphical client provided by Java to connect to any application using JMX. Moreover, we provide a command line interface that interacts with the components via JMX.

**Persistence:** relies on a relational database for storing the information used by the gateway. Information such as peering arrangements and templates for virtual machines are persistently stored in the database. The database is provided by the Apache Derby project[2], which is a database implemented entirely in Java.

**Scheduler**: the component here represented by the Scheduler comprises other components, namely the resource provisioning policy, the peering directory, request redirection and the enactment module. The scheduler interacts with the Virtual Machine Manager in order to create, start or stop VMs to fulfill the requirements of the scheduled requests. The scheduler also maintains the availability information obtained by the VM Manager.

### Virtual Machine Manager (VMM)

The Virtual Machine Manager (VMM) is the link between the gateway and the resources. As described previously, the gateway does not share physical resources directly, but relies on virtualization technology for abstracting them.

[2] http://db.apache.org/derby

Hence, the actual resources used by the gateway are virtual machines. The VMM relies on a Virtual Infrastructure Engine (VIE), for managing VMs on a set of physical resources. Typically, VIEs are able to create and stop VMs on a physical cluster. At present, we use Open Nebula[3] as a VIE. Open Nebula allows the user to submit VMs on physical machines using different kinds of hypervisors, such as XEN[4]. Xen is a hypervisor that allows running several operating systems on a same host concurrently. A hypervisor gives privileged access to the hardware for the guest operating systems. The host can control and limit the usage of certain resources by the guests, such as memory or CPU. In addition, the VMM manages the deployment of VMs on Grids and IaaS providers.

**Virtual Machine template**: Open Nebula's terminology is used to explain the idea of templates for VMs. A template is analogous to a computer's configuration, and contains a description for a virtual machine with the following information:

- Number of cores or processors to be assigned to the VM;
- The amount of memory required by the VM;
- The kernel used to boot the VM's operating system;
- The disk image containing the VM's file system; and
- Optionally, the price of using a VM over one hour.

That information is static, *i.e.* described once and reusable, and is provided by the gateway administrator at the set-up time of the infrastructure. The administrator can update, add, and delete templates at any time. In addition, each gateway of the InterGrid network has to agree on the templates in order to provide the same configuration on each site.

To deploy an instance of a given template, a descriptor is generated from the information in the template. The descriptor contains the same fields as the

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

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

template and additional information related to a specific VM instance. The additional fields are:

- The disk image that contains the file system for the VM;
- The address of the physical machine that hosts the VM;
- The network configuration of the VM; and
- For deploying on an IaaS provider, the required information on the remote infrastructure, such as account information for the provider.

Before starting an instance, the network configuration and the address of the host are given by the scheduler. The scheduler allocates a MAC address and/or an IP address for that instance. The disk image field is specified by the template but can be modified in the descriptor. To deploy several instances of the same template in parallel, each instance uses a temporary copy of the disk image specified by the template. Hence, the descriptor contains the path to the copied disk image.

The fields of the descriptor can be different for deploying a VM on an IaaS provider. For example, the network information is not mandatory for using Amazon as EC2 automatically assigns a public IP to the instances. In addition, EC2 creates copies of the disk image seamlessly for running several instances in parallel. Before running an instance, the disk image must be uploaded to Amazon EC2 thus ensuring that the template has its corresponding disk image.

**Virtual Machine Template Directory**, the gateway works with a repository of VM templates. That is, templates can be registered to the repository by the gateway administrator. In this way, a user can request instances of VMs whose template is registered on the gateway's repository. In addition, the gateway administrator has to upload the images to Amazon if the gateway uses the cloud as a resource provider. The user currently cannot submit her own template or disk images to the gateway.

**Virtual Machine Manager** allows the gateway to submit and deploy VMs on a physical infrastructure. It also interacts with a VIE to create and stop virtual

machines on a physical cluster. The implementation of the VMM is generic in order to connect with different VIEs. We have developed VMMs for Open Nebula, Amazon EC2, and Grid'5000. The connection with Open Nebula consists in using the Java client[5], to submit and stop VMs and in transforming our VM template to the format recognized by Open Nebula. Open Nebula runs as a daemon service on a master node, hence the VMM works as a remote user of Open Nebula.

We have also implemented a connector to deploy VMs on IaaS providers. For the moment, only Amazon is supported. The connector is a wrapper for the command line tool provided by Amazon. In addition, we have developed an emulated VIE for testing and debugging purposes. The VMM for Grid'5000 is also a wrapper for the command line tools (*i.e.* OAR scheduler and Kadeploy virtualization). The emulator provides a list of fake machines, where we can set the number of cores for hosting VMs. Figure 4 shows the interaction between the gateway, the template directory, and the VMM.

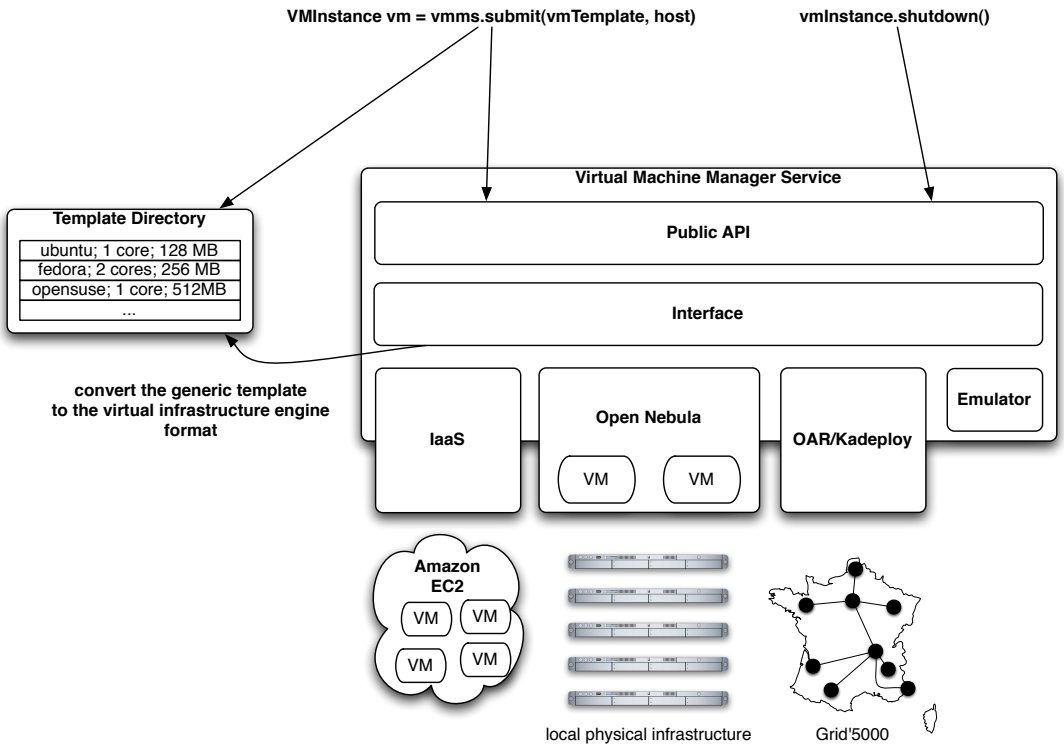


**Figure 4: Design and interactions of the Virtual Machine Manager.**

[5] http://opennebula.org/doku.php?id=ecosystem#java_api

### Distributed Virtual Environment Manager (DVE-Manager)

A DVE Manager interacts with the IGG by making requests for VMs and querying their status. The DVE Manager requests VMS from the gateway on behalf of the user application it represents.

When the reservation starts, the DVE manager obtains the list of requested VMs from the gateway. This list contains a tuple of public IP/private IP for each VM, which the DVE Manager uses to access the VMs (with SSH tunnels). With EC2, VMs have a public IP hence the DVE can access to the VMs directly without tunnels. Then, the DVE Manager handles the deployment of the user's application. We have evaluated the system with a bag-of-tasks application without dependencies between tasks.

### InterGrid Gateway at Runtime

Figure 5 shows the main interactions between InterGrid's components. The user first requests VMs, the request is handle by a command line interface. The user must specify which VM template she wants to use; and can also specify the number of VM instances, the ready time for her reservation, the deadline, the walltime, and the address of an alternative gateway. The client returns an identifier for the submitted request from the gateway. Next, the user starts a DVE manager with the returned identifier (or a list of identifiers) and its application as parameters. The application is described by a text file where each line is one task to execute on a remote VM. The task is indeed the command line to run with SSH. The DVE Manager waits until the request has been scheduled or refused. The local gateway tries to obtain resources from the underlying VIEs. When that is not possible, the local gateway starts a negotiation with remote gateways in order to fulfill the request. When a gateway can fulfill the request, *i.e.* can schedule the VMs, it sends the access information to connect to the assigned VMs to the requester gateway. Once the requester gateway has collected all the VM access information, this information is made available for the DVE Manager. Finally, the DVE Manager configures the VMs, set-up SSH tunnels, and executes the tasks on the VMs. In future work, we want to improve the description of

applications to allow file-transfer, dependencies between tasks, and VM configuration.

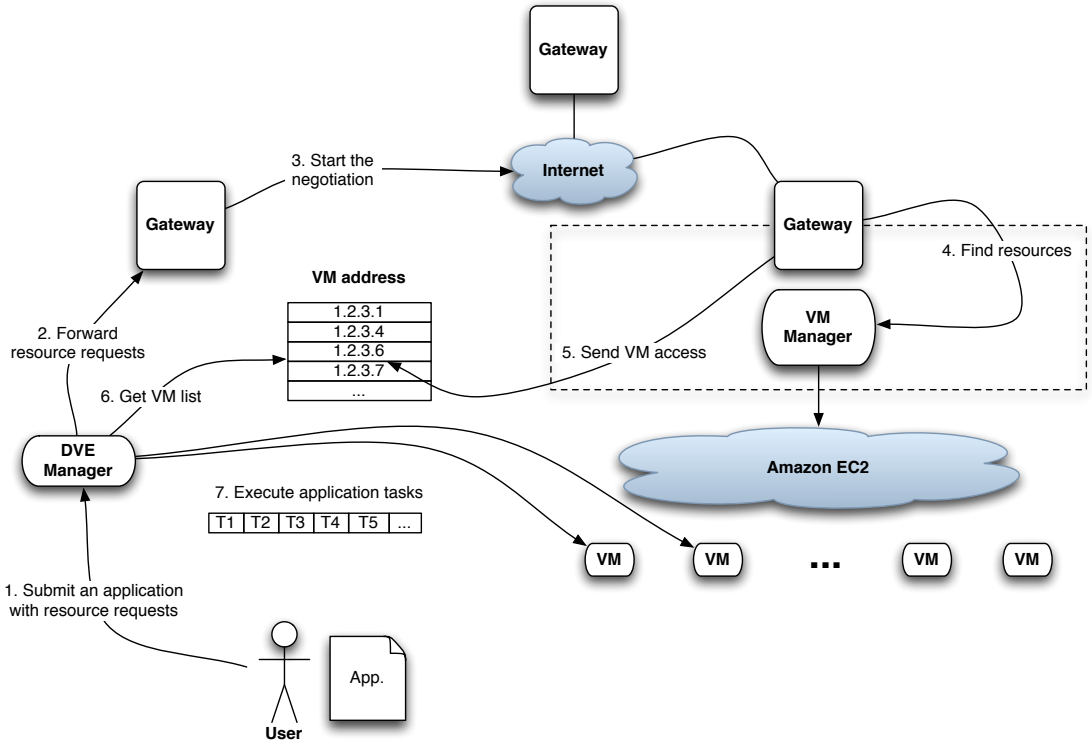


**Figure 5: The main interactions among the InterGrid components.**

Under the peering policy considered in this work, each gateway's scheduler uses conservative backfilling to schedule the requests. When the scheduler cannot start a request immediately using local resources, then a redirection algorithm will:

1) Contact the remote gateways and ask for offers containing the earliest start time at which they would be able to serve the request, if it were redirected.

2) For each offer received, check whether the request start time proposed by a peering gateway is than that given by local resources. That being the case, the request is redirected; otherwise, the algorithm will check the next offer.

3) If the request start time given by local resources is better than those proposed by remote gateways, then the algorithm will schedule the request locally.

This strategy has been previously proposed and evaluated in a smaller environment including a local cluster and a cloud computing provider [9].

## *Experiments*

This section describes two experiments. The first experiment evaluates the performance of allocation decisions by measuring how the gateways manage load via peering arrangements. The second experiment considers the effectiveness of InterGrid for deploying a bag-of-tasks application on Cloud providers.

### Peering Arrangements

This experiment uses the French experimental Grid platform, Grid'5000, as scenario and testbed. Grid'5000 comprises nine sites geographically distributed in France, currently featuring 4792 cores.

Each gateway created in this experiment represents one site of Grid'5000; the gateway runs on that site. To prevent gateways from interfering on real users of Grid'5000, we use the emulated VMM, which instantiates fictitious VMs. The number of emulated hosts is the number of real cores available on each site. Figure 6 illustrates the Grid'5000 sites and the evaluated gateway configurations.

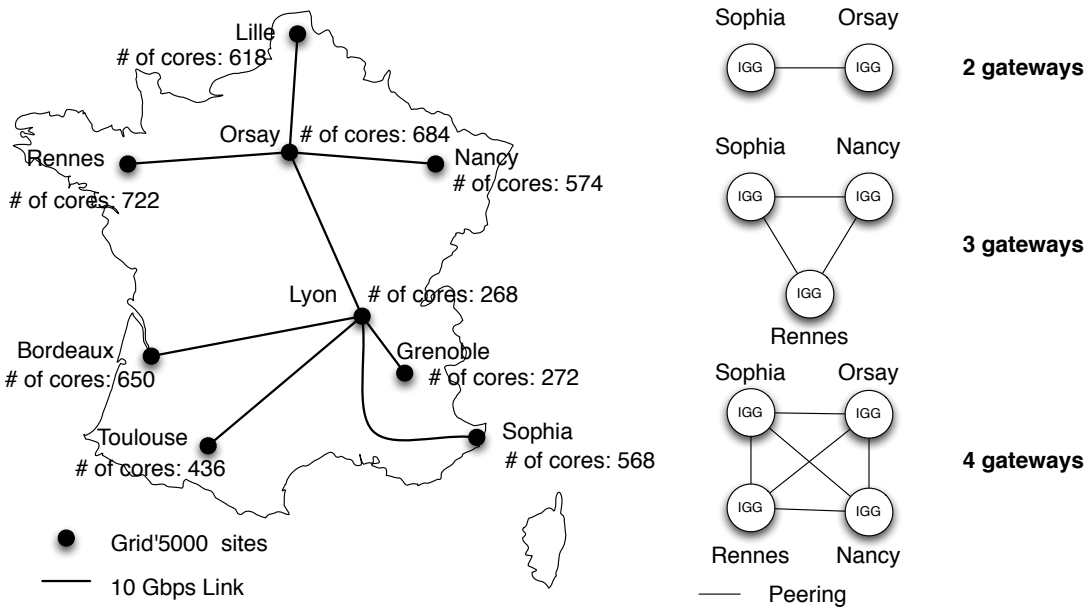


**Figure 6: InterGrid testbed over Grid'5000.**

The site's workloads are generated using Lublin and Feitelson's model [10], here referred to as Lublin99. Lublin99 is configured to generate one-day-long workloads; the maximum number of VMs required by generated requests is the number of cores in the site. To generate different workloads, the mean number of virtual machines required by a request (specified in $\log_2$) is set to $\log_2 m - umed$, where $m$ is the maximum number of virtual machines allowed in system. We randomly vary *umed* from 1.5 to 3.5. In addition, to simulate a burst of request arrivals and heavy loads, thus stretching the system, we multiply the inter-arrival time by 0.1.

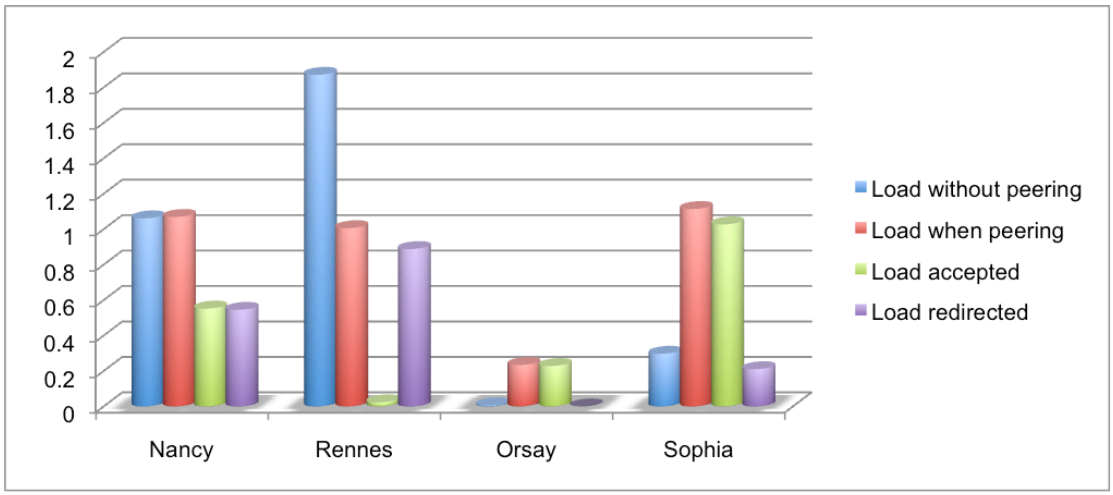


**Figure 7: Load characteristics under the four-gateway scenario.**

Figure 7 shows the load characteristics under the four-gateway scenario. The blue bars indicate the load of each site when they are not interconnected; the red bars show the load when gateways redirect requests to one another; the green bars correspond to the amount of load accepted by each gateway from other gateways; and the purple bars represent the amount of load redirected. The results show that the policy used by gateways balances the load across sites, making the it tend to 1. Rennes, a site with heavy load, benefits from peering with other gateways as a great share of its load is redirected to other sites.

Table 1 presents the job slowdown improvement caused by the interconnection of gateways. Overall, the job slowdown is improved by the interconnection. Sites with the heaviest loads (*i.e.* Rennes and Nancy) have better improvements. The job slowdown of sites with lower loads is in fact worsened; however, this impact

is minimized as the number of gateways increases, which leads to the conclusion that sites with light load suffer a smaller impact as the number of interconnected gateways increases. The experiments demonstrate that peering is overall beneficial to interconnected sites; benefits derive from load balancing and overall job slowdown improvement.

**Table 1: Job slowdown improvement under different gateway configurations.**

| Site | 2 Gateways | 3 Gateways | 4 Gateways |
|---|---|---|---|
| Orsay | -0.00006 | N/A | 0.00010 |
| Nancy | N/A | 3.95780 | 4.30864 |
| Rennes | N/A | 7.84217 | 12.82736 |
| Sophia | 0.20168 | -6.12047 | -3.12708 |

## Deploying a Bag-of-Tasks Application

This experiment considers Evolutionary Multi-criterion Optimization (EMO) [11], a bag-of-tasks application for solving optimization problems using a multi-objective evolutionary algorithm. Evolutionary algorithms are a class of population-based meta-heuristics exploiting the concept of population evolution to find solutions to optimization problems. The optimal solution is found by using an iterative process that evolves the collection of individuals in order to improve the quality of the solution. Each task is an EMO process that explores a different set of populations.

**Table 2: Experimental results with 1 and 2 gateways using resources from Amazon EC2.**

| Number of VMs | 1 Gateway (time in seconds) | 2 Gateways Each gateway provides 1/2 of the VMs (time in seconds) |
|---|---|---|
| **5** | 4,780 | - |
| **10** | 3,017 | 3,177 |
| **15** | 2,407 | - |
| **20** | 2,108 | 2,070 |

Figure 8 shows the testbed for running the experiments. We carry out one experiment in two steps. First, we evaluate the execution time of EMO using a single gateway. Then, we force InterGrid to provide resources from two gateways. In this case, we limit the number of available cores for running VMs,

and the DVE Manager submits two requests. For 10 VMs, both gateways are limited to nine cores and the DVE Manager sends two requests for five VMs each. Next, for 20 VMs gateways set the limit to 20 cores and the DVE Manager requests 10 VMs twice. The two gateways use resources from EC2. The requests demand a small EC2 instance running Windows 2003 Server. Table 2 reports the results of both experiments. The experiments show that the execution time of the bag-of-tasks application does not suffer important performance degradations with one or two gateways.

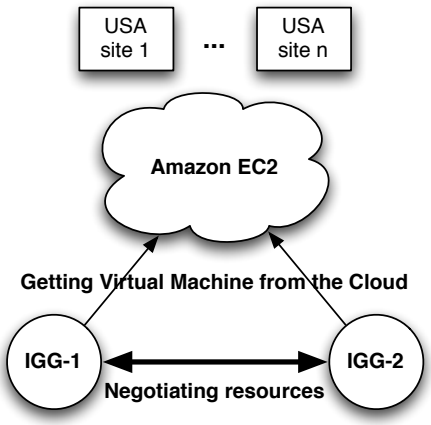


**Figure 8: Testbed used to run EMO on a Cloud Computing provider.**

## *Conclusion*

This article presented a system, the InterGrid, for deploying applications on multiple-site execution environments. The system relies on gateways inspired by the peering agreements between ISPs. We described the virtualization technology used by InterGrid to abstract physical infrastructures. Then, we showed the motivation and the realization of the InterGrid system.

As the management of the local resource is based on virtual infrastructure engines, we detail how the gateway interacts with these engines to control virtual machines. In addition, the InterGrid can instantiate VMs on Amazon EC2 and deploy system images on Grids, such as Grid'5000.

Experimental results emulating a real Grid showed the performance of allocation decisions made by gateways. We also presented a runtime scenario for deploying

a parallel evolutionary algorithm for solving optimization problems. With that application, we validated the realization of the InterGrid by comparing the runtime execution of the application when obtaining resources from one gateway at a time and from two gateways, which were deploying VMs on Amazon EC2.

In future work, we plan to improve the VM template directory to allow users to submit their own VMs and to synchronize the available VMs between gateways. In addition, the security aspects have not been addressed in this work because they are handled at the operating system and network levels, it would be interesting to address those aspects at the InterGrid level.

**Acknowledgments**

We thank Mohsen Amini for helping in the system implementation. This work is supported by research grants from the Australian Research Council and Australian Department of Innovation, Industry, Science and Research. Marcos' PhD research is partially supported by NICTA. Some experiments were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER, several universities, and other funding bodies (see https://www.grid5000.fr).

## *References*

[1] Enabling Grids for E-sciencE (EGEE) project, http://public.eu-egee.org (2005).

[2] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard. "Grid'5000: a large scale and highly reconfigurable grid experimental testbed". In 6th IEEE/ACM International Workshop on Grid Computing, 2005.

[3] Grid Interoperability Now Community Group (GIN-CG), http://forge.ogf.org/sf/projects/gin (2006).

[4] X. Zhu, D. Young, B. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, et al. 1000 Islands: Integrated Capacity and Workload Management for the Next Generation Data Center. In Autonomic Computing, 2008. ICAC'08. International Conference on, pages 172–181, 2008.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[6] M. D. de Assunção, R. Buyya, S. Venugopal, InterGrid: A case for internetworking islands of Grids, Concurrency and Computation: Practice and Experience (CCPE) 20 (8) (2008) 997–1024.

[7] K. Keahey, I. Foster, T. Freeman, X. Zhang, Virtual workspaces: Achieving quality of service and quality of life in the Grids, Scientific Programming 13 (4) (2006) 265–275.

[8] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, S. E. Sprenkle, Dynamic virtual clusters in a Grid site manager, in: 12th IEEE International Symposium on High Performance Distributed Computing (HPDC 2003), IEEE Computer Society, Washington, DC, USA, 2003, p. 90.

[9] M. D. de Assunção, A. di Costanzo and R. Buyya. Evaluating the Cost-Benefit of Using Cloud Computing to Extend the Capacity of Clusters, In Proceedings of the International Symposium on High Performance Distributed Computing (HPDC 2009), Munich, Germany, 11-13 Jun. 2009.

[10] U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. Journal of Parallel and Distributed Computing, 63 (11) (2008) 1105–1122.

[11] M. Kirley, and R. Stewart, "Multiobjective evolutionary algorithms on complex networks", in Proc. of the Fourth International Conference on Evolutionary Multi-Criterion Optimization, Lecture Notes Computer Science 4403, Springer Berlin, Heidelberg, 2007, pp. 81–95.

**Alexandre di Costanzo** is a research fellow at the University of Melbourne. His research interests are distributed computing and especially grid computing. Di Costanzo has a PhD in computer science from the University of Nice Sophia Antipolis, France. Contact him at adc@csse.unimelb.edu.au.

**Marcos Dias de Assunção** is a PhD candidate at the University of Melbourne, Australia. His PhD thesis is on peering and resource allocation across Grids. The current topics of his interest include Grid scheduling, virtual machines, and network virtualization. Contact him at marcosd@csse.unimelb.edu.au

**Rajkumar Buyya** is an Associate Professor and Reader of Computer Science and Software Engineering; and Director of the Grid Computing and Distributed

Systems (GRIDS) Laboratory at the University of Melbourne, Australia. He also serves as CEO of Manjrasoft, Australia. Contact him at raj@csse.unimelb.edu.au.