Harnessing Cloud Technologies for a Virtualized Distributed Computing Infrastructure

Article *in* IEEE Internet Computing · November 2009

DOI: 10.1109/MIC.2009.108 · Source: IEEE Xplore

3 authors, including:

Marcos Assuncao
École de Technologie Supérieure
104 PUBLICATIONS **3,914** CITATIONS

SEE PROFILE

Rajkumar Buyya
University of Melbourne
1,023 PUBLICATIONS **102,896** CITATIONS

SEE PROFILE

Harnessing Cloud Technologies for a Virtualized Distributed Computing Infrastructure

The InterGrid system aims to provide an execution environment for running applications on top of interconnected infrastructures. The system uses virtual machines as building blocks to construct execution environments that span multiple computing sites. Such environments can be extended to operate on cloud infrastructures, such as Amazon EC2. This article provides an abstract view of the proposed architecture and its implementation; experiments show the scalability of an InterGrid-managed infrastructure and how the system can benefit from using the cloud.

Alexandre di Costanzo, Marcos Dias de Assunção, and Rajkumar Buyya
University of Melbourne

Over the past decade, the distributed computing field has been characterized by large-scale grid deployments, such as the Enabling Grids for E-science project (EGEE; http://public.eu-egee.org) and Grid'5000.[1] Such grids have given the research community an unprecedented number of resources, which it's used for various scientific endeavors. Several efforts have been made to enable grids to interoperate — for instance, by providing standard components and adapters for secure job submissions, data transfers, and information queries (see http://forge.ogf.org/sf/projects/gin). Despite these efforts, software and hardware heterogeneity has contributed to the increasing complexity inherent in deploying applications on these infrastructures. Moreover, recent advances in virtualization technologies[2] have led to the emergence of commercial infrastructure providers, a concept known as *cloud computing*.[3] Handling distributed applications' ever-growing demands 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[4] inspired by the peering agreements established between ISPs, through which they agree to allow traffic into each others' networks. Here, we look at the realization of this architecture, which we call the Inter-

 1089-7801/09/$26.00 © 2009 IEEE

Related Work in Interconnecting Infrastructures

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

Systems for virtualizing a physical infrastructure are also available. Rubén S. Montero and colleagues investigated the deployment of custom execution environments using OpenNebula.[2] The authors investigated the overhead of two distinct models for starting virtual machines and adding them to an execution environment. A.J. Rubio-Montero and colleagues used GridWay to deploy virtual machines on a Globus Grid[3]; jobs are encapsulated as virtual machines. The authors showed that the overhead of starting a virtual machine is small for the application evaluated.

Researchers have investigated several load-sharing mechanisms in the distributed systems realm. Alexandru Iosup and colleagues proposed a matchmaking mechanism for enabling resource sharing across computational grids.[4] Sonesh Surana and colleagues addressed the load balancing in distributed hash-table-based peer-to-peer networks.[5]

References

1. P. Ruth, P. McGachey, and D. Xu, "VioCluster: Virtualization for Dynamic Computational Domain," *Proc. IEEE Int'l Conf. Cluster Computing* (Cluster 05), IEEE Press, 2005, pp. 1–10.
2. R.S. Montero, E. Huedo, and I.M. Llorente, "Dynamic Deployment of Custom Execution Environments in Grids, *Proc. 2nd Int'l Conf. Advanced Engineering Computing and Applications in Sciences* (ADVCOMP 08), IEEE CS Press, 2008, pp. 33–38.
3. A.J. Rubio-Montero et al., "Management of Virtual Machines on Globus Grids using GridWay," *Proc. IEEE Int'l Parallel and Distributed Processing Symp.* (IPDPS 07), IEEE CS Press, 2007, pp. 1–7.
4. A. Iosup et al., "Inter-Operating Grids through Delegated Matchmaking, *Proc. 2007 ACM/IEEE Conf. Supercomputing* (SC 07), ACM Press, 2007, pp. 1–12.
5. S. Surana et al., "Load Balancing in Dynamic Structured Peer-to-Peer Systems," *Performance Evaluation*, vol. 63, no. 3, 2006, pp. 217–240.

Grid. Our architecture relies on *InterGrid gateways* (IGGs) that mediate access to participating grids' resources. It also aims to tackle hardware and software heterogeneity within grids. Using virtualization technology can ease the deployment of applications spanning multiple grids because it allows for resource control in a contained manner. In this way, resources allocated by one grid to another can help deploy virtual machines (VMs), which also let InterGrid use resources from cloud computing providers.

Background and Context

With the recent advances in multicore technologies, virtualization has become more adoptable and provides solutions for interconnecting heterogeneous distributed infrastructures. (See the "Related Work in Interconnecting Infrastructures" sidebar for more about this topic.)

Virtualization Technology and Infrastructure as a Service

VM technologies' increasing ubiquity has enabled users to create customized environments atop physical infrastructure and has facilitated the emergence of business models such as cloud computing. VMs' use has several benefits:

- server consolidation, which lets system administrators place the workloads of several underutilized servers in fewer machines;
- the ability to create VMs to run legacy code without interfering with other applications' APIs;
- improved security through the creation of sandboxes for running applications with questionable reliability; and
- performance isolation, letting providers offer some guarantees and better quality of service to customers' applications.

Existing VM-based resource management systems can manage a cluster of computers within a site, allowing users to create virtual workspaces[5] or clusters.[6] Such systems can bind resources to virtual clusters or workspaces according to a user's demand. They commonly provide an interface through which users can allocate VMs and configure them with a chosen operating system and software. These resource managers, or *virtual infrastructure engines* (VIEs), let users create customized virtual clusters by using shares of the physical machines available at the site.

Virtualization technology minimizes some security concerns inherent to the sharing of resources among multiple computing sites. Indeed, we use virtualization software to realize the InterGrid architecture because existing cluster resource managers relying on VMs can give us the building blocks — such as availability information — required to create virtual execution environments. In addition, relying on

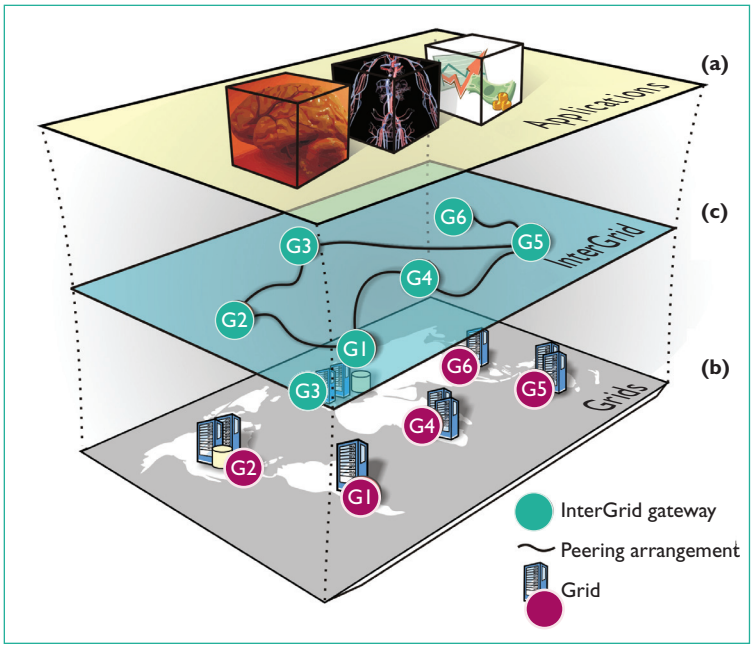

Figure 1. InterGrid software layers. InterGrid aims to provide a software system that lets users create execution environments for (a) various applications on top of (b) physical infrastructure that participating grids provide. (c) Peering arrangements established between gateways enable the allocation of resources from multiple grids to fulfill the execution environments' requirements.

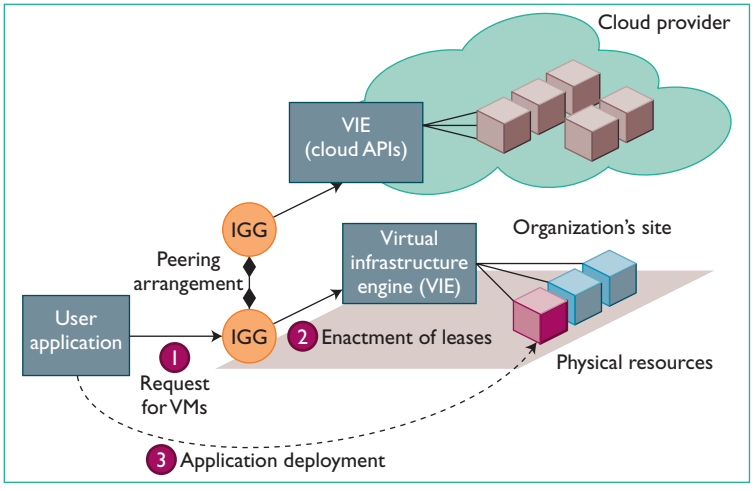

Figure 2. Application deployment. An InterGrid gateway (IGG) allocates resources from one organization's local cluster and interacts with another IGG that can allocate resources from a cloud computing provider.

VMs eases the deployment of execution environments on multiple computing sites; user applications can have better control over the software installed on the allocated resources without compromising 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 Internet-accessible services: software-as-a-service (SaaS), platform-as-a-service (PaaS), and infrastructure-as-a-service (IaaS). Here, we consider only IaaS, which aims to provide computing resources or storage as a service to users. One major player in cloud computing is Amazon's Elastic Compute Cloud (EC2; http://aws.amazon.com/ec2/), which comprises several data centers worldwide. Amazon EC2 lets users deploy VMs on-demand on Amazon's infrastructure and pay only for the computing, storage, and network resources they use.

InterGrid Concepts

In this article, we'll examine the InterGrid's realization, but let's first look at its overall architecture; more details are available in our previous work.[4]

Figure 1 depicts the scenario InterGrid considers. InterGrid aims to provide a software system that lets users create execution environments for various applications on top of the physical infrastructure participating grids provide. Peering arrangements established between gateways enables the allocation of resources from multiple grids to fulfill the execution environments' requirements.

A grid has predefined peering arrangements with other grids, which IGGs manage and through which IGGs coordinate the use of InterGrid's resources. An IGG is aware of the peering terms this grid has with other grids, selects suitable grids that can provide the required resources, and replies to requests from other IGGs. Request redirection policies determine which peering grid InterGrid selects to process a request and a price for which that grid will perform it. An IGG can also allocate resources from a cloud provider. Figure 2 illustrates a scenario in which an IGG allocates resources from an organization's local cluster to deploy applications. Under peak demand, 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 propose a system that creates a virtual environment to help users deploy their applications. These applications use the resources InterGrid allocates and provides as a *distributed virtual*

environment (DVE) — that is, a network of VMs that runs isolated from other networks of VMs. A component called the *DVE manager* performs resource allocation and management on the application's behalf.

InterGrid Realization

To realize our system, we've implemented the IGG in Java; Figure 3 depicts its main components.

The *communication module* receives messages from other entities and delivers them to the components registered as listeners for those types of messages. It also lets components communicate with other entities in the system by providing the functionality for sending messages. Message-passing helps make gateways loosely coupled and able to build more failure-tolerant communication protocols. In addition, senders and receivers are decoupled, making the entire system more resilient to traffic bursts. One central component, the *post office*, handles all the communication module's functionalities. When the post office receives an incoming message, it associates the message to a thread and then forwards it to all listeners. Threads are provided by a *thread-pool*, and if the pool is empty, the post office puts messages in a queue to wait for available threads. Listeners are message handlers and deal with messages. All listeners are notified when a new message arrives and can individually decide to process the message. Because messages are asynchronous and sent or served in parallel, no order or delivery guarantees are possible by default. Rather, communicating components are responsible for handling those properties. For instance, the *scheduler* (described later) uses unique message identifiers to manage request negotiations.

InterGrid uses Java Management Extensions (JMX) to perform *management and monitoring*. JMX is a standard API for managing and monitoring resources such as Java applications. It also includes remote secure access, so a remote program can interact with a running application for management purposes. Via JMX, the gateway exports management operations such as configuring peering, connecting or disconnecting to another gateway, shutting down the gateway, and managing the VM manager. All these operations are accessible via JConsole, a graphical Java client that lets an administrator (for instance) connect to any application using JMX. Moreover, we provide a command-line interface that interacts with components via JMX.

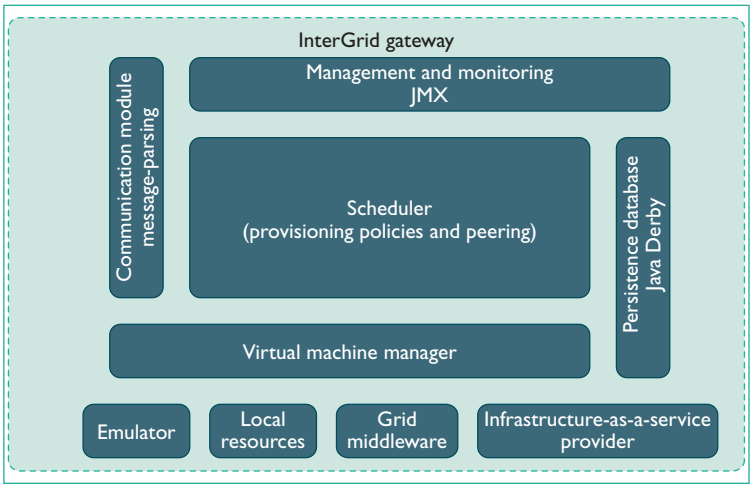

Figure 3. InterGrid gateway components. The core component is the scheduler, which implements the provisioning policies and peering with other gateways. The communication component provides an asynchronous message-passing mechanism, and received messages are handled in parallel by a thread-pool.

Persistence relies on a relational database for storing the information the gateway uses. Information such as peering arrangements and VM templates are persistently stored in the database, which the Apache Derby project (http://db.apache.org/derby) provides. This database is implemented entirely in Java.

The scheduler component comprises several other components — namely, the resource provisioning policy, the peering directory, request redirection, and the enactment module. The scheduler interacts with the *virtual machine manager* (VMM) to create, start, or stop VMs to fulfill scheduled requests' requirements. The scheduler also maintains the availability information the VMM obtains.

Virtual Machine Manager

The VMM is the link between the gateway and the resources. As we described, the gateway doesn't share physical resources directly but relies on virtualization technology for abstracting them. Hence, the actual resources it uses are VMs. The VMM relies on a VIE to manage VMs on a set of physical resources. VMM implementation is generic so that it can connect with different VIEs. Typically, VIEs can create and stop VMs on a physical cluster. We've developed VMMs for OpenNebula, Amazon EC2, and Grid'5000. At present, we use OpenNebula (www.opennebula.org) as a VIE for deploying VMs on a local infrastructure. The connection

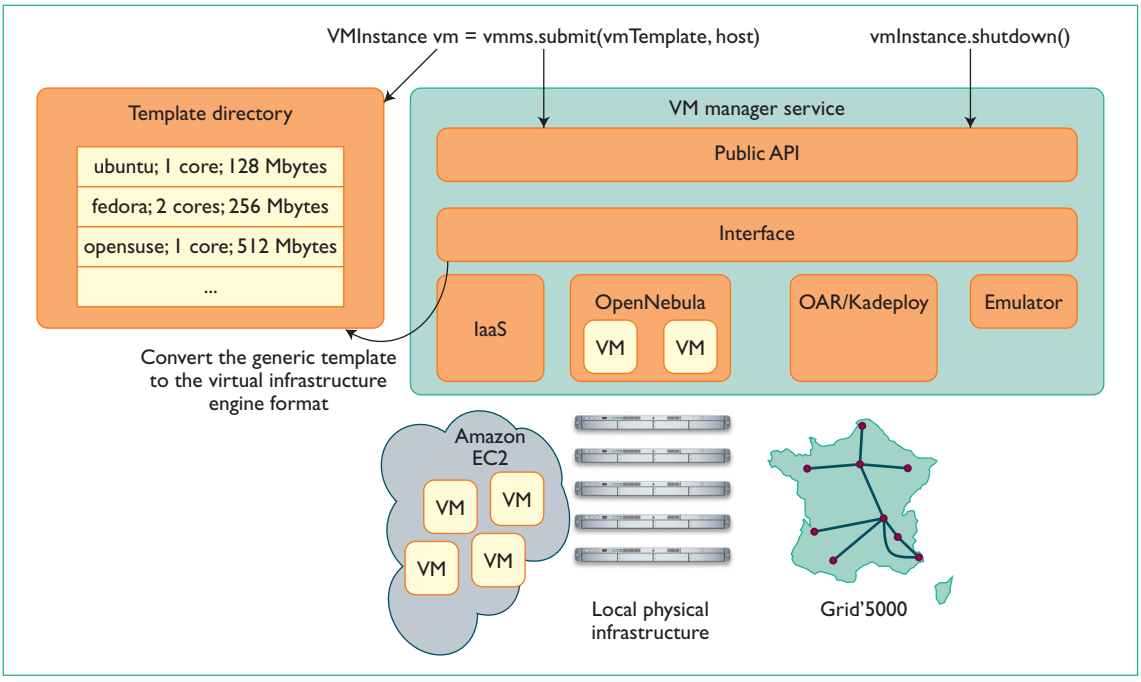

Figure 4. Design and interactions within the virtual machine (VM) manager. The manager provides a public API for submitting and controlling VMs. This API is abstract, and we provide a few implementations for deploying VMs on different types of resources.

with OpenNebula uses the Java client (http://opennebula.org/doku.php?id=ecosystem#java_api) to submit and stop VMs and transform our VM template (described later) to the format OpenNebula recognizes. OpenNebula runs as a daemon service on a master node, so the VMM works as a remote user. It lets users submit VMs on physical machines using different kinds of hypervisors, such as Xen (www.xen.org), which enables running several operating systems on the same host concurrently. A hypervisor gives guest operating systems privileged access to the hardware. The host can control and limit guests' use of certain resources, such as memory or CPU.

The VMM also manages VM deployment on grids and IaaS providers. We've implemented a connector to deploy VMs on IaaS providers, but so far, InterGrid supports only Amazon EC2. The connector is a wrapper for the command-line tool Amazon provides. The VMM for Grid'5000 is also a wrapper for its command-line tools (that is, the OAR scheduler and Kadeploy virtualization). In addition, we've developed an emulated VIE for testing and debugging purposes. The emulator provides a list of fake machines where we can set the number of cores for hosting VMs.

To deploy a VM, the VMM needs a description of it, or *template*. Figure 4 shows the interaction between the gateway, the *template directory*, and the VMM.

VM template. OpenNebula's terminology can help explain the idea of templates for VMs. A template is analogous to a computer's configuration and contains a description for a VM with the following information:

- the number of cores or processors to be assigned to the VM;
- the amount of memory the VM requires;
- 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.

This information is static — that is, it's described once and reusable; the gateway administrator provides it when the infrastructure is set up. The administrator can update, add, and delete templates at any time. In addition, each gateway in the InterGrid network must agree on the templates to provide the same configuration on each site.

To deploy an instance of a given template, the VMM generates a descriptor from the informa-

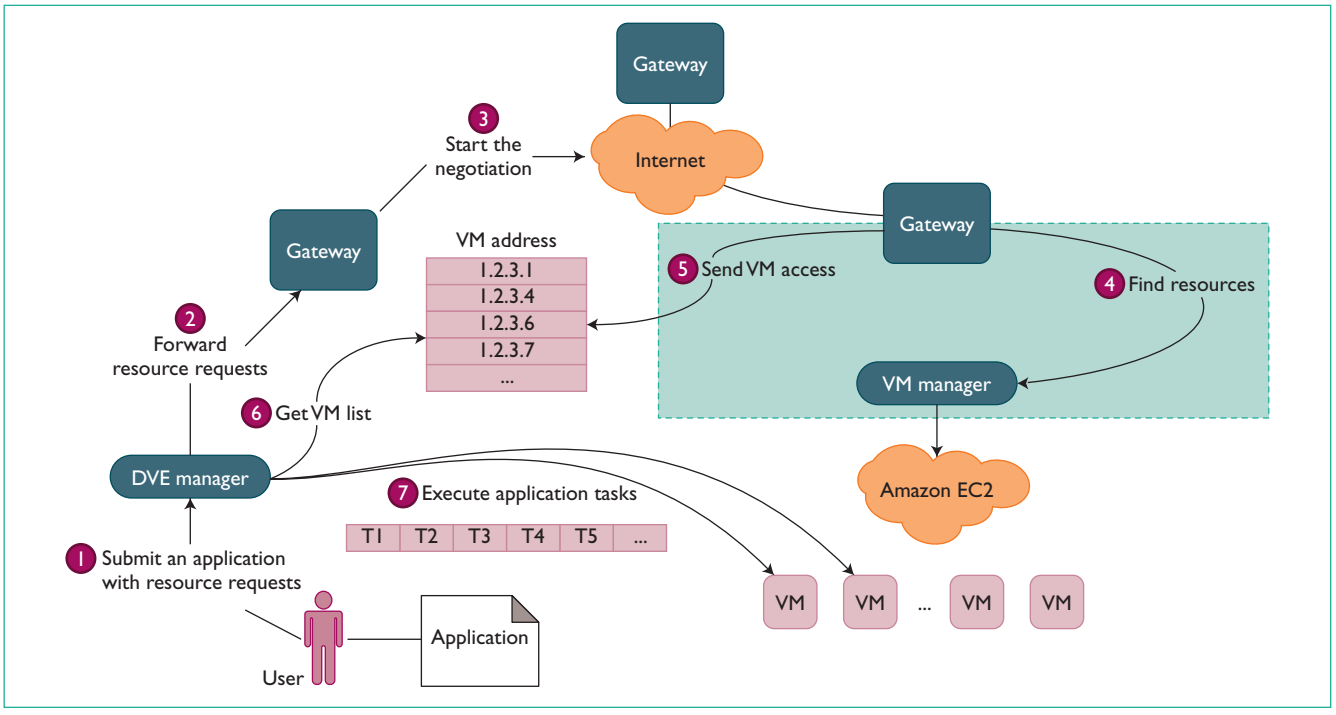

Figure 5. The main interactions among InterGrid components. On the user's behalf, the distributed virtual environment (DVE) manager requests resources to a gateway. The gateway then tries to serve the request locally or starts negotiating with other gateway to fulfill it. Once a gateway can serve the request, the virtual machine manager (VMM) deploys the resources on top of the infrastructure and returns the access information about the VM to the requesting gateway. Finally, the DVE manager fetches the VM access from the gateway and deploys the user's application.

tion in the template. This descriptor contains the same fields as the template and additional information related to a specific VM instance, such as

- the disk image that contains the VM's file system;
- the address of the physical machine hosting the VM;
- the VM's network configuration; and
- the required information on the remote infrastructure (for deployment on an IaaS provider), such as account information for the provider.

Before starting an instance, the scheduler gives the network configuration and the host's address; it then allocates MAC and IP addresses for that instance. The template specifies the disk image field, but this field 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 the template has specified. Hence, the descriptor contains the path to the copied disk image.

The descriptor's fields are different for deploying a VM on an IaaS provider. Network information isn't mandatory for using Amazon because EC2 automatically assigns a public IP to the instances. In addition, EC2 creates copies of the disk image seamlessly to run several instances in parallel. Before running an instance, an InterGrid administrator must upload the disk image to Amazon EC2, thus ensuring that the template has its corresponding disk image.

VM template directory. The IGG works with a repository of VM templates — that is, the gateway administrator can register templates to the repository to let users find and request instances of specific VMs. In addition, the gateway administrator must upload the images to Amazon if the gateway uses the cloud as a resource provider. Users currently can't submit their own templates or disk images to the gateway.

Distributed Virtual Environment 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.

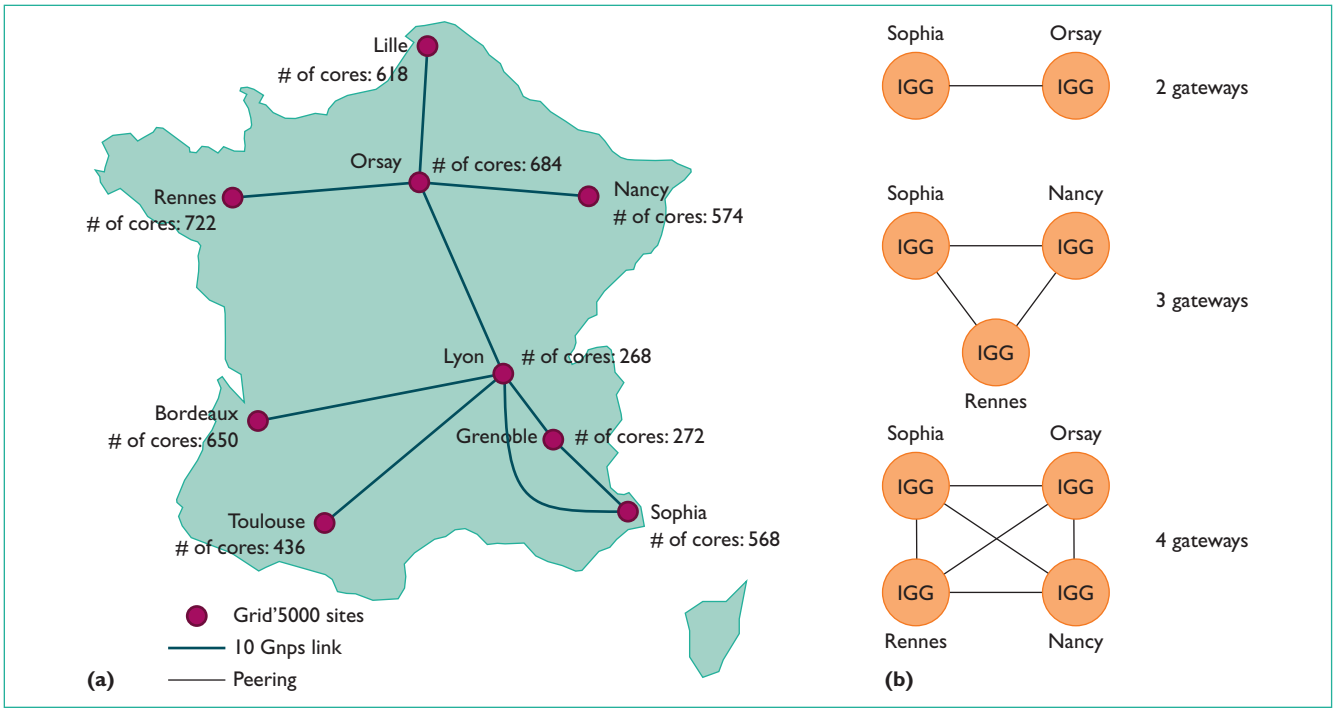

Figure 6. InterGrid testbed over Grid'5000. We can see the Grid'5000 sites as well as the gateway configurations we evaluated.

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 Secure Shell [SSH] tunnels). With EC2, VMs have a public IP, so the DVE can access the VMs directly without tunnels. Then, the DVE manager deploys the user's application.

InterGrid Gateway at Runtime

Figure 5 shows the main interactions between InterGrid's components. When the user first requests a VM, a command-line interface handles the request. Users must specify which VM template they want to use; they can also specify the number of VM instances, the ready time for the reservation, the deadline, the walltime (that is, the time the user estimates the job will take), and the address for 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 via a text file in which each line is one task to execute on a remote VM. (The task is indeed the command line that runs with SSH.) The DVE manager waits until InterGrid has scheduled or refused the request. The local gateway tries to obtain resources from the underlying VIEs. When this isn't possible, the local gateway starts a negotiation with any remote gateways to fulfill the request. When a gateway can fulfill the request — that is, can schedule the VMs — it sends the access information for connecting to the assigned VM to the requester gateway. Once this gateway has collected all the VM access information, it makes it available for the DVE manager. Finally, the DVE manager configures the VM, sets up SSH tunnels, and executes the tasks on the VM. In future work, we want to improve the description of applications to allow file transfer, dependencies between tasks, and VM configuration.

Under the peering policy we consider in this work, each gateway's scheduler uses conservative backfilling to schedule requests. When the scheduler can't start a request immediately using local resources, then a redirection algorithm will take the following steps:

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

Table 1. Job slowdown improvement under different gateway configurations.

Site	Two gateways	Three gateways	Four 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

request start time a peering gateway proposes is that given by local resources. This being the case, the algorithm redirects the request; otherwise, the algorithm will check the next offer.

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

We previously proposed and evaluated this strategy in a smaller environment that included a local cluster and a cloud computing provider.[7]

Experiments

We conducted two experiments to evaluate our InterGrid architecture. The first evaluates the performance of allocation decisions by measuring how the IGGs manage load via peering arrangements. The second considers InterGrid's effectiveness in deploying a bag-of-tasks application on cloud providers.

Peering Arrangements

For our testing, we used the French experimental grid platform, Grid'5000, as both a scenario and a testbed. Grid'5000 comprises nine sites geographically distributed across France, and currently features 4,792 cores.

Each gateway created in this experiment represents one Grid'5000 site; the gateway runs on that site. To prevent gateways from interfering with real Grid'5000 users, we used 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.

We generated the site's workloads using Uri Lublin and Dror G. Feitelson's model,[8] which we refer to here as Lublin99. We configured Lublin99 to generate one-day-long workloads; the maximum number of VMs that generated requests require is the number of cores in the site. To generate different workloads, we set the mean number of VMs that a request requires (specified in $\log_2$) to $\log_2 m - umed$, where m is the maximum number of VMs allowed in the system. We randomly varied $umed$ from 1.5 to 3.5. In addition, to simulate a burst of request arrivals and heavy loads, thus stretching the system, we multiplied the interarrival time by 0.1.

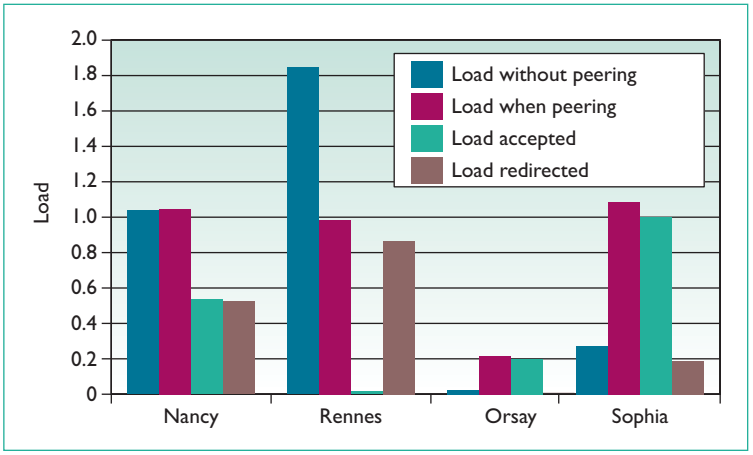

Figure 7. Load characteristics under the four-gateway scenario. The teal bars indicate each site's load when they aren't interconnected; the magenta bars show the load when gateways redirect requests to one another; the green bars correspond to the amount of load each gateway accepts from other gateways; and the brown bars represent the amount of load redirected.

Figure 7 shows the load characteristics under the four-gateway scenario. The teal bars indicate each site's load when they aren't interconnected; the magenta bars show the load when gateways redirect requests to one another; the green bars correspond to the amount of load each gateway accepts from other gateways; and the brown bars represent the amount of load redirected. The results show that the policy the gateways use balances the load across sites, making it tend to 1. Rennes, a site with heavy load, benefits from peering with other gateways as the gateway redirects a great share of its load to other sites.

Table 1 presents the job slowdown improvement resulting from gateway interconnection. Overall, the interconnection improves job slow-

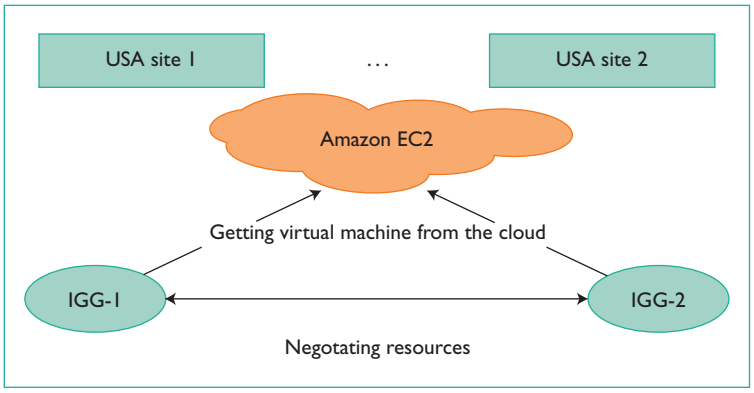

Figure 8. Testbed used to run Evolutionary Multi-Criterion Optimization on a cloud computing provider. The testbed is composed of two InterGrid gateways (IGGs), each using resources from Amazon EC2.

Table 2. Experimental results with one and two gateways using resources from Amazon EC2.

Number of virtual machines (VMs)	One gateway (seconds)	Two gateways* (seconds)
5	4,780	–
10	3,017	3,177
15	2,407	–
20	2,108	2,070

*Each gateway provides half of the VMs

down — for example, sites with the heaviest loads (that is, Rennes and Nancy) have better improvements. However, the job slowdown of sites with lower loads gets worse; as the number of gateways increases, though, this impact is minimized, which leads to the conclusion that sites with light loads suffer a smaller impact when more interconnected gateways are present. This experiment demonstrates that peering is overall beneficial to interconnected sites — these benefits derive from load balancing and overall job slowdown improvement.

Deploying a Bag-of-Tasks Application

For our second experiment, we considered *Evolutionary Multi-Criterion Optimization* (EMO),[9] a bag-of-tasks application for solving optimization problems using a multi-objective evolutionary algorithm. Evolutionary algorithms are a class of population-based metaheuristics that exploit the concept of population evolution to find solutions to optimization problems. They can find the optimal solution using an iterative process that evolves the collection of individuals to improve the solution's quality. Each task is an EMO process that explores a different set of populations.

Figure 8 shows the testbed for running the experiment. We carried out each test in two steps. First, we evaluated EMO's execution time using a single gateway, and then we forced InterGrid to provide resources from two gateways. In this case, we limited the number of available cores for running VMs, and the DVE manager submitted two requests. For 10 VMs, we limited both gateways to five cores, and the DVE manager sent two requests for five VMs each. Next, for 20 VMs, we set the limit to 10 cores, and the DVE manager requested 10 VMs twice. The two gateways used resources from Amazon EC2 — the requests demanded a small EC2 instance running Windows Server 2003. Table 2 reports both steps' results. The execution time of the bag-of-tasks application doesn't suffer important performance degradations with one or two gateways.

Our experiments with InterGrid have shown that it can balance load between distributed sites and have validated that a bag-of-tasks application can run on distributed sites using VMs. We currently provide a minimal gateway that lets resource providers interconnect sites and deploy VMs on different kinds of infrastructures, such as local clusters, Amazon EC2, and Grid'5000.

In future work, we plan to improve the VM template directory to let users submit their own VMs and synchronize the available VMs between gateways. In addition, although we haven't addressed security aspects in this work because they're handled at the operating system and network levels, it would be interesting to address those concerns 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 the Australian Department of Innovation, Industry, Science, and Research. Marcos Dias de Assunção's PhD research is partially supported by National Information and Communications Technology Australia. We carried out some experiments using the Grid'5000 experimental testbed being developed under the INRIA Aladdin development action with support from CNRS, RENATER (Centre national de la recherche scientifique, Réseau National de télécommunications pour

la Technologie l'Enseignement et la Recherche), several universities, and other funding bodies (see https://www.grid5000.fr). The experiments run on Amazon EC2 are supported by an Amazon Web Services Research Grant.

References

1. F. Cappello et al., "Grid'5000: A Large-Scale and Highly Reconfigurable Grid Experimental Testbed," *Int'l J. High Performance Computing Applications*, vol. 20, no. 4, 2006, pp. 481–494.
2. X. Zhu et al., "1000 Islands: Integrated Capacity and Workload Management for the Next Generation Data Center," *Proc. Int'l Conf. Autonomic Computing* (ICAC 08), IEEE CS Press, 2008, pp. 172–181.
3. M. Armbrust et al., *Above the Clouds: A Berkeley View of Cloud Computing*, tech. report UCB/EECS-2009-28, EECS Dept., Univ. of California, Berkeley, Feb. 2009.
4. M. Dias de Assunção, R. Buyya, and S. Venugopal, "InterGrid: A Case for Internetworking Islands of Grids," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 8, 2008, pp. 997–1024.
5. K. Keahey et al., "Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grids," *Scientific Programming*, vol. 13, no. 4, 2006, pp. 265–275.
6. J.S. Chase et al., "Dynamic Virtual Clusters in a Grid Site Manager," *Proc. 12th IEEE Int'l Symp. High Performance Distributed Computing* (HPDC 03), IEEE CS Press, 2003, p. 90.
7. M. Dias de Assunção, A. di Costanzo, and R. Buyya, "Evaluating the Cost-Benefit of Using Cloud Computing to Extend the Capacity of Clusters," *Proc. Int'l Symp. High Performance Distributed Computing* (HPDC 09), ACM Press, 2009, pp. 141–150.
8. U. Lublin and D.G. Feitelson, "The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs," *J. Parallel and Distributed Computing*, vol. 63, no. 11, 2008, pp. 1105–1122.
9. M. Kirley and R. Stewart, "Multiobjective Evolutionary Algorithms on Complex Networks," *Proc. 4th Int'l Conf. Evolutionary Multi-Criterion Optimization*, LNCS 4403, Springer, 2007, pp. 81–95.

Alexandre di Costanzo is a research fellow at the University of Melbourne. His research interests are in distributed and 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. His PhD thesis is on peering and resource allocation across grids, and his interests include grid scheduling, virtual machines, and network virtualization. Dias de Assunção has a PhD in computer science from the University of Melbourne. Contact him at marcosd@csse.unimelb.edu.au.

Rajkumar Buyya is an associate professor and reader of computer science and software engineering as well as the director of the Grid Computing and Distributed Systems (GRIDS) Laboratory at the University of Melbourne. He also serves as CEO of Manjrasoft. Contact him at raj@csse.unimelb.edu.au.