Sky Computing

Infrastructure-as-a-service (IaaS) cloud computing is revolutionizing how we approach computing. Compute resource consumers can eliminate the expense inherent in acquiring, managing, and operating IT infrastructure and instead lease resources on a pay-as-you-go basis. IT infrastructure providers can exploit economies of scale to mitigate the cost of buying and operating resources and avoid the complexity required to manage multiple customer-specific environments and applications. The authors describe the context in which cloud computing arose, discuss its current strengths and shortcomings, and point to an emerging computing pattern it enables that they call sky computing.

Katarzyna Keahey
University of Chicago

Maurício Tsugawa, Andréa Matsunaga, and José A.B. Fortes
University of Florida

The idea of using remote resources for regular computing work and on a large scale was first manifested with *grid computing*, which is based on the assumption that control over how resources are used stays with the site, reflecting local software and policy choices. However, these choices aren't always useful to remote users who might need a different operating system or login access instead of a batch scheduler interface to a site. Reconciling those choices between multiple user groups proved to be complex, time-consuming, and expensive. In retrospect, leaving control to individual sites was a pragmatic choice that enabled very fast adoption of a radically transformative technology, but also led to a "local maximum" beyond which grid computing found it hard to scale.

Infrastructure-as-a-service (IaaS) *cloud computing* represents a fundamental change from the grid computing assumption: when a remote user "leases" a resource, the service provider turns control of that resource over to the user. This change was enabled when a free and efficient virtualization solution, the Xen hypervisor (www.xen.org), became available. Before virtualization, turning over control to users was fraught with danger because users could easily subvert a site. But virtualization isolates the leased resource from the site in a secure way,

1089-7801/09/$26.00 © 2009 IEEE

mitigating this danger. In turn, this ability to give users control over a remote resource lets us develop tools — such as those the Amazon Elastic Compute Cloud (EC2; http://aws.amazon.com/ec2/) uses — or academic projects — such as Nimbus, an EC2-compatible open source IaaS implementation (http://workspace.globus.org) — that let users carve out their own custom "sites."

Here, we consider another change, brought about by cloud computing's emergence as a mainstream platform. Previously, site owners couldn't trust a remote resource because they had no control over its configuration. Now that clouds let users control remote resources, however, this concern is no longer an issue. Combining the ability to trust remote sites with a trusted networking environment, we can now lay a virtual site over distributed resources. Because such dynamically provisioned distributed domains are built over several clouds, we call this kind of computing *sky computing*. In grids, the interaction among different sites consisted of interaction among different isolated domains, but the trust relationships within sky computing are the same as those within a traditional nondistributed site, simplifying how remote resources interact.

Applications that can leverage a sky computing platform range from multitier, seasonal e-commerce or Web-server-like systems in which different components are on different clouds, to CAD systems with different tools and data hosted in different clouds when design activities are under way, to distributed, event-based alert systems that discover and integrate information or data patterns across databases hosted in different clouds. Here, we discuss the challenges in building sky computing platforms, present the ingredients for a working system, and discuss a potential sky computing application.

Interoperability and Service Levels

IaaS exposes an API that lets a client programmatically provision and securely take ownership of customized computer infrastructure for an agreed-upon time period. To create a reliable sky computing platform, we need such capability to be uniformly available across providers. Users must be able to easily compare offerings from different providers — choosing between qualities of service (for example, availability, reliability, or performance at different price points) — and move from one provider to another. Let's look at where standards are needed to make IaaS cloud computing a fungible resource.

To easily choose between providers, a client needs to move from one to another without significantly altering its mode of usage. The first obstacle to such movement is *image compatibility*. Not only do various virtual machine (VM) implementations, such as VMware (www.vmware.com), Xen, or Kernel-based Virtual Machine (KVM; www.linux-kvm.org), use different disk formats, but moving images between different deployments of one implementation can also be a challenge. For example, Xen images used with paravirtualization often rely on a specific, externally provided kernel — if different providers supply different kernels, an image that works with one might fail with another due to integration issues. One way to deal with this issue is to simply publish the relevant information and work only with image-compatible providers. Although this solves the problem, it also restricts end users' choices.

Another reason why images — even compliant ones — might work with one provider but not another is *contextualization compatibility*. VM instances deployed based on a shared image must be customized with information (typically at startup) that lets end users employ each instance in a specific context — for example, to allow login to certain parties. However, *contextualization* can be provided in a variety of ways (Amazon EC2, for example, provides it via the EC2 metadata structure accessed over an internal network, whereas the Open Virtualization Format [OVF] specification[1] advocates providing it by mounting a file system), and no agreement on standard methods exists, so each provider might do it differently. Even slight differences, especially if not clearly documented, contribute to the difficulty of moving from one provider to another.

The best way to deal with these incompatibilities so far has been to define environments not in terms of their implementation but via an abstraction — a virtual appliance[2] — from which service providers can derive any implementation. This approach, which commercial providers support via existing tools, could resolve both image incompatibilities (images are generated) and provider incompatibilities (images are generated and customized for each provider).

Standards in both areas, as well as more structured user data management, would lower the barrier to interoperability.

Finally, standards are needed at an *API-level compliance* among different cloud providers. So far, emergent APIs tend toward compatibility — semantically, at least — because they all implement a roughly similar set of functions: deploy and terminate environments. Where differences are emerging, and where it's useful for them to emerge, is with service-level agreements (SLAs), which describe such wide-ranging qualities as differentiated service security levels, allocated resources, availability, and price. The more providers can explicitly define these qualities (as opposed to implicitly, as with an obscure option embedded into a deployment command), the easier comparing SLAs among different IaaS providers will be.

An inherent difficulty exists, however, in explicitly describing virtualized environments, especially in terms of the performance and resource quota offered. Today, commercial providers define SLA qualities as "instances" — for example, a VM deployed on specific hardware with certain performance specifications, which might include nebulous terms such as "high bandwidth I/O." These definitions hide the fact that a virtualization's implementation version and configuration, and how it interacts with specific hardware, might significantly affect how much of an instance is in fact available to the VM and with what trade-offs.[3] One possibility for creating a comparison base is to define a usefully comprehensive set of benchmarks that providers can publish to give users an idea of which performance factors are relevant (much as car manufacturers publish such benchmarks today). Differences in network services make it difficult for VMs deployed in distinct providers to establish communication (for example, network and security policies, or private networks). A promising solution to enable intercloud communication is to use user-level overlay networks.

At present, there isn't much to guide and structure such SLA development. Although existing specifications, such as OVF[1] and the Web Services Agreement Specification (WS-Agreement),[4] can provide some guidance, in practice, emerging de facto standards (such as Amazon EC2) guide and influence that development.

Creating a Sky Computing Domain

Several building blocks underpin the creation of a sky environment. While leveraging cloud computing, we can in principle trust the configuration of remote resources, which will typically be connected via untrusted WANs. Furthermore, they won't be configured to recognize and trust each other. So, we need to connect them to a trusted networking domain and configure explicit trust and configuration relationships between them. In short, we must provide an end-user environment that represents a uniform abstraction — such as a virtual cluster or a virtual site — independent of any particular cloud provider and that can be instantiated dynamically. We next examine the mechanisms that can accomplish this.

Creating a Trusted Networking Environment

Network connectivity is particularly challenging for both users and providers. It's difficult to offer APIs that reconfigure the network infrastructure to adjust to users' needs without giving them privileged access to core network equipment — something providers wouldn't do owing to obvious security risks. Without network APIs, establishing communication among resources in distinct providers is difficult for users.

Deploying a "virtual cluster" spanning resources in different providers faces challenges in terms of network connectivity, performance, and management:

- *Connectivity.* Resources in independently administered clouds are subject to different connectivity constraints due to packet filtering and network address translations; techniques to overcome such limitations are necessary. Due to sky computing's dynamic, distributed nature, reconfiguring core network equipment isn't practical because it requires human intervention in each provider. Network researchers have developed many overlay networks to address the connectivity problem involving resources in multiple sites, including NAT-aware network libraries and APIs, virtual networks (VNs), and peer-to-peer (P2P) systems.
- *Performance.* Overlay network processing negatively affects performance. To minimize performance degradation, compute resources should avoid overlay network processing when it's not necessary. For example, requir-

ing overlay network processing in every node (as with P2P systems) slows down communication among nodes on the same LAN segment. In addition, overlay network processing is CPU intensive and can take valuable compute cycles from applications. A detailed study of overlay network processing performance is available elsewhere.[5]

- *Service levels.* Sky computing requires on-demand creation of mutually isolated networks over heterogeneous resources (compute nodes and network equipment) distributed across distant geographical locations and under different administrative domains. In terms of SLAs, this has security as well as performance implications. Core network routers and other devices are designed for a single administrative domain, and management coordination is very difficult in multisite scenarios. Overlay networks must be easily deployable and agnostic with respect to network equipment vendors.

To address these issues and provide connectivity across different providers at low performance cost, we developed the Virtual Networks (ViNe) networking overlay.[6] ViNe offers end-to-end connectivity among nodes on the overlay, even if they're in private networks or guarded by firewalls. We architected ViNe to support multiple, mutually isolated VNs, which providers can dynamically configure and manage, thus offering users a well-defined security level. In performance terms, ViNe can offer throughputs greater than 800 Mbps with submillisecond latency, and can handle most traffic crossing LAN boundaries as well as Gigabit Ethernet traffic with low overhead.

ViNe is a user-level network routing software, which creates overlay networks using the Internet infrastructure. A machine running ViNe software becomes a ViNe router (VR), working as a gateway to overlay networks for machines connected to the same LAN segment. We recommend delegating overlay network processing to a specific machine when deploying ViNe so that the additional network processing doesn't steal compute cycles from compute nodes, a scenario that can occur if all nodes become VRs.

ViNe offers flexibility in deployment as exemplified in the following scenarios.

ViNe-enabled providers. Providers deploy a VR in each LAN segment. The ability to dynamically and programmatically configure ViNe overlays lets providers offer APIs for virtual networking without compromising the physical network infrastructure configuration. The cost for a provider is one dedicated machine (which could be a VM) per LAN segment and can be a small fraction of the network cost charged to users. IaaS providers offer VN services in this case.

End-user clusters. In the absence of ViNe services from providers, users can enable ViNe as an additional VM that they start and configure to connect different cloud providers. This user-deployed VR would handle the traffic crossing the cluster nodes' LAN boundaries. ViNe's cost in this case is an additional VM per user.

Isolated VMs. A VR can't be used as a gateway by machines that don't belong to the same LAN segment. In this case, every isolated VM (or a physical machine, such as the user's client machine) must become a VR. ViNe's cost is the additional network processing that compute nodes perform, which can take compute cycles from applications.

Dynamic Configuration and Trust

When we deploy a single appliance with a specific provider, we rely on basic security and contextualization measures this provider has implemented to integrate the appliance into a provider-specific networking and security context (for example, to let the appliance owner log in). However, when we deal with a group of appliances, potentially deployed across different providers, configuration and security relationships are more complex and require provider-independent methods to establish a security and configuration context.

In earlier work,[7] we describe a *context broker service* that dynamically establishes a security and configuration context exchange between several distributed appliances. Orchestrating this exchange relies on the collaboration of three parties:

- *IaaS providers*, who provide generic contextualization methods that securely deliver to deployed appliances the means of contacting a context broker and authenticating themselves to it as members of a specific context

Table 1. Service-level agreement and instances at each cloud provider.

	University of Chicago (UC)	University of Florida (UF)	Purdue University (PU)
Xen version	3.1.0	3.1.0	3.0.3
Guest kernel	2.6.18-x86_64	2.6.18-i686	2.6.16-i686
Nimbus version	2.2	2.1	2.1
CPU architecture	AMD Opteron 248	Intel Xeon Prestonia	Intel Xeon Irwindale
CPU clock	2.2 GHz	2.4 GHz	2.8 GHz
CPU cache	1 Mbyte	512 Kbytes	2 Mbytes
Virtual CPUs per node	2	2	2
Memory	3.5 Gbytes	3.5 Gbytes	1.5 Gbytes
Networking	Public	Private	Public

(and possibly also as individual appliances). End users provide context information via a simple generic schema and method that's the same for every appliance used with this provider. Adopting this simple schema lets every provider deliver basic context information to every appliance.

- *Appliance providers*, who provide methods that let an appliance supply information to and receive it from a context broker and integrate information conveyed by templates describing application-specific roles. Appliances can integrate the information using any configuration method from any appliance provider. This information in the templates is application-specific and potentially different from appliance to appliance, but the templates themselves are uniform, and any context broker can process them.
- *Deployment orchestrators* (*context brokers*), who provide generic methods of security context establishment and information exchange based on information the appliance templates provide.

A typical contextualization process works as follows. Before a user deploys appliances, he or she registers a context object with a context broker. This object is identified by an identifier and a secret. The IaaS provider securely conveys the identifier and secret (along with ways to contact the context broker) on deployment. This gives the appliance a way to authenticate itself to the context broker, which can then orchestrate security context establishment as well as information exchange between all appliances in the context (external sources can provide additional security and configuration information to the security broker).

Defining this exchange in terms of such roles lets any appliance contextualize with any provider (or across providers). For example, using the Nimbus toolkit (http://workspace.globus.org) implementation of a context broker, we could dynamically deploy clusters of appliances on Nimbus's Science Clouds (including multiple Science Cloud providers) as well as Amazon EC2.[7]

Building Metaclouds

Next, let's look at how we can exploit resource availability across different Science Clouds (http://workspace.globus.org/clouds/), offering different SLAs, to construct a sky environment: a virtual cluster large enough to support an application execution. Rather than simply selecting the provider with the largest available resources, we select IaaS allocations from a few different providers and build a sky environment on top of those allocations using the ViNe network overlay and the Nimbus context exchange tools.

The Science Clouds testbed comprises multiple IaaS providers configured in the academic space and providing different SLAs to users; Science Cloud providers grant access to resources to scientific projects, free of charge and upon request. Apart from providing a platform on which scientific applications explore cloud computing, the Science Clouds testbed creates a laboratory in which different IaaS providers use compatible technologies to provide offerings, letting us experiment with sky computing.

Our sky computing study uses resources on three sites: University of Chicago (UC), University of Florida (UF), and Purdue University (PU). All sites use the same virtualization implementation (Xen), and although the versions and kernels differ slightly, VM images are portable across sites. All sites use Nimbus so that VM images are contextualization-compliant across those sites.

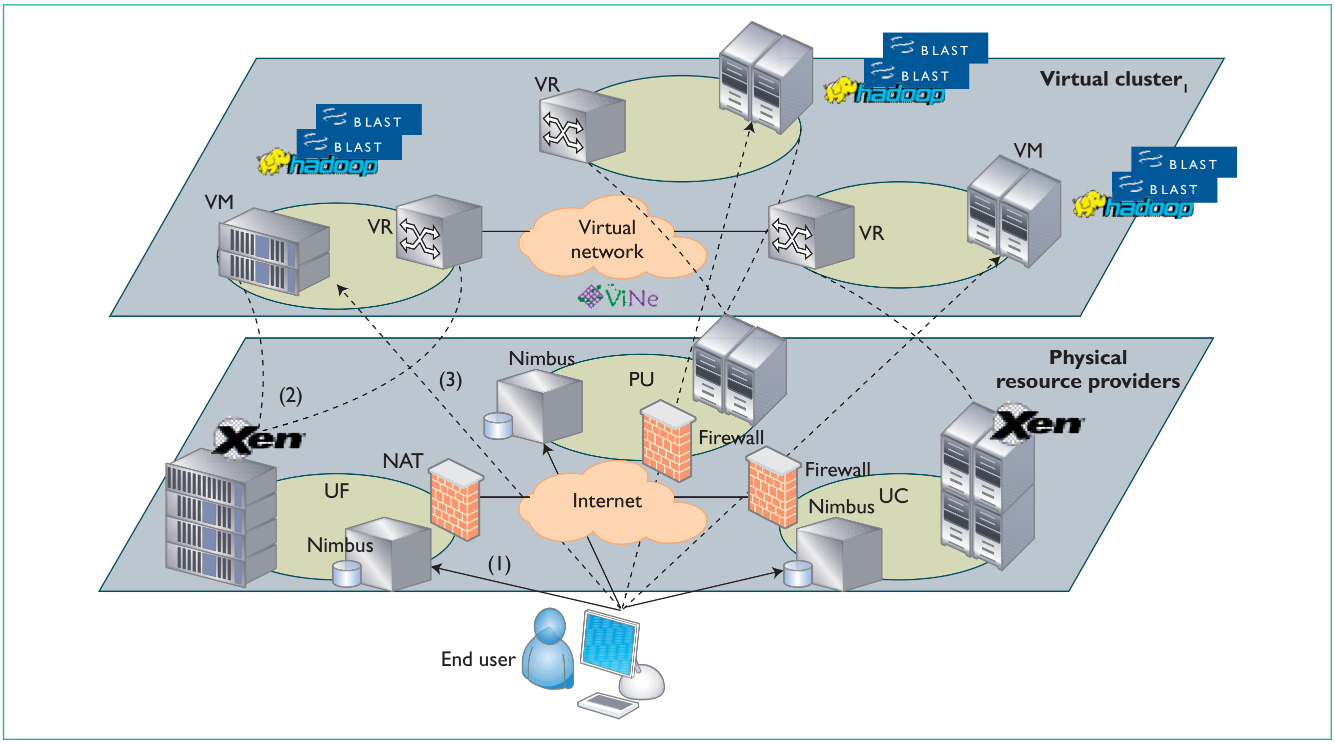

Figure 1. A virtual cluster interconnected with ViNe. An end user employs the Nimbus client, contextualization service, and images available in the marketplace. We can see (1) preparation, (2) deployment, and (3) usage.

Consequently, the sites are also API-compliant, but, as Table 1 shows, they offer different SLAs. Although all sites offer an "immediate lease," the provided instances (defined in terms of CPU, memory, and so on) are different. More significantly from a usability viewpoint, the UC and PU clouds provide public IP leases to the deployed VMs, whereas UF doesn't.

To construct a sky virtual cluster over the testbed we just described, a user with access to the Science Clouds testbed takes the following steps (see Figure 1):

- *Preparation*. Obtain a Xen VM image configured to support an environment the application requires as well as the ViNe VM image (the ViNe image is available from the Science Clouds Marketplace). Make sure both images are contextualized (that is, capable of providing and integrating context information). The user must upload both images to each provider site.
- *Deployment*. Start a ViNe VM in each site (the ViNe VMs provide virtual routers for the network overlay). In addition, start the desired number of compute VMs at each provider site. The contextualized images are configured to automatically (securely) contact the context broker to provide appropriate networking and security information and adjust network routes to use VRs to reach nodes crossing site boundaries. The configuration exchange includes VMs on different provider sites so that all VMs can behave as a single virtual cluster.
- *Usage*. Upload inputs and start the desired application (typically, by simply logging into the virtual cluster and using a command-line interface).

To experiment with the scalability of virtual clusters deployed in different settings, we configured two clusters: a Hadoop cluster, using the Hadoop MapReduce framework, version 0.16.2 (http://hadoop.apache.org), and a message passing interface (MPI) cluster using MPICH2 version 1.0.7 (www.mcs.anl.gov/research/projects/mpich2/). We used each virtual cluster to run parallel versions of the Basic Local Alignment Search Tool (Blast), a popular bioinformatics application that searches for, aligns, and ranks nucleotide or protein sequences that are similar to those in an existing database of known sequences. We configured the Hadoop cluster with Blast version 2.2.18 (http://blast.ncbi.nlm.nih.gov) and the MPI cluster with the publicly

Table 2. Normalized single processor performance at each site.*

	University of Chigaco (UC)	University of Florida (UF)	Purdue University (PU)
Sequential execution time	36 hours and 23 minutes	43 hours and 06 minutes	34 hours and 49 minutes
Normalization factor	1.184	1	1.24

*Measured as the Blast sequential time at the University of Florida divided by the Blast sequential time at each site

available mpiBlast version 1.5.0beta1 (www.mpiblast.org). Both versions have master-slave structures with low communication-to-computation ratios. The master coordinates sequence distribution among workers, monitoring their health and combining the output. The runs used in the evaluation consisted of executing blastx of 960 sequences averaging 1,116.82 nucleotides per sequence against a 2007 nonredundant (NR) protein sequence database from the US National Center for Biotechnology Information (NCBI) in 1 fragment (3.5 Gbytes of total data.)

We deployed the two virtual clusters in two settings: on the UF cloud only (one-site experiment) and on all three sites using the same number of processors (three-site experiment). For three-site experiments, we balanced the number of hosts in each site executing Blast — that is, one host in each site, two hosts in each site, and so on, up to five hosts in each site. (Choosing random numbers of nodes from different sites would, in effect, weigh the three-site experiment's performance toward comparing the UF site and the site with the most processors).

The SLAs expressed as instances from each metacloud provider (as described in Table 1) are different (PU instances outperform UC instances, which outperform UF instances), which makes it difficult to compare providers. To establish a comparison base between the SLAs each provider offers, we used the performance of the sequential execution on a UF processor of the Blast job described earlier to define a normalized performance benchmark (see Table 2): 1 UC processor is equivalent to 1.184 UF processors, whereas 1 PU processor is equivalent to 1.24 UF processors. For example, an experiment with 10 UF processors, 10 UC processors, and 10 PU processors should provide the performance of a cluster with 34.24 UF processors. We used these factors to normalize the number of processors, as Figure 2 shows.

Figure 2 shows the speedup Blast execution on various numbers of testbed processors in different deployment settings versus the execution on one processor at UF. A sequential execution on one UF processor resource that took 43 hours and 6 minutes was reduced to 1 hour and 42 minutes using Hadoop on 15 instances (30 processors) of the UF cloud, a 25.4-fold speedup. It was reduced to 1 hour and 29 minutes using Hadoop on five instances in each of the three sites (30 processors), a 29-fold speedup. Overall, the performance difference between a virtual cluster deployed in a single cloud provider and a virtual cluster deployed in three distinct cloud providers interconnected across a WAN through a VN is minimal for Blast executed with either Hadoop or MPI. Also, comparison with "ideal" performance (assuming perfect parallelization — that is, where *N* CPU clusters would provide *N*-fold speedup relative to sequential execution) shows that the application parallelizes well.

In the data presented, we refer only to the VMs used to create the application platform and not to those additional ones used to run VRs. Running those routers (one per site) constitutes an additional cost in resource usage. This cost is relatively small and depends on network traffic, as detailed elsewhere.[5] We can further amortize this cost by sharing the router with other cloud users (the provider could offer it as another service) or running it in one of the compute nodes.

Our experiments aimed to study the feasibility of executing a parallel application across multiple cloud providers. In this context, our two main objectives were to demonstrate that end users can deploy a sky computing environment with full control, and that the environment performs well enough to execute a real-world application. We've successfully combined open source and readily available cloud (Nimbus toolkit) and VN (ViNe) technologies to let users launch virtual clusters with nodes that are automatically configured and connected through overlays. The observed impact of network virtualization overheads was low, and we could sustain the performance of a single-site cluster using a cluster across three sites. This illustrates sky computing's potential in that even when the necessary resources are unavail-

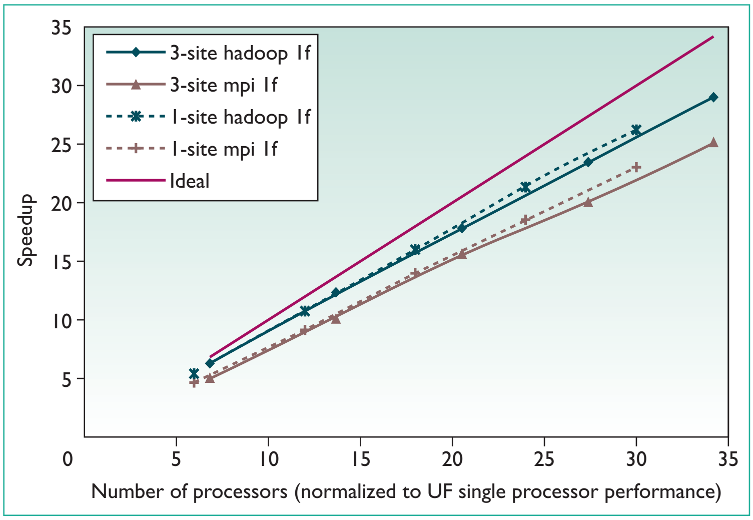

Figure 2. Speedup curves. These curves compare the performance of virtual clusters running Hadoop and message passing interface (MPI) versions of Blast in a single site (University of Florida [UF]) and in three different sites (UF, University of Chicago, and Purdue University) for different (normalized) numbers of processors. Both versions show good speedup, with the three-site performance being comparable to that obtained using a single site.

able in a single cloud, we can use multiple clouds to get the required computation power.

Our work brought forward several interesting features. First, leveraging VMs' isolation property let us create a distributed environment that we couldn't have created otherwise: ViNe deployment requires root privileges on remote resources, which would have made its dynamic deployment on those sites hard if not impossible. The combination of contextualization technology and network overlays further lets users dynamically create a sky environment, which is easy to use. The complexity moves to lower infrastructure layers that provide middleware and prepare images. However, these actions can occur once, be amortized over many different uses, and, most importantly, don't affect the end user's work.

Our work also exposed some shortcomings of current cloud computing systems — namely, the difficulty of comparing offerings coming from different providers. For our example, we used an application-specific benchmark to establish a comparison base. However, for more general usage, a standardized set of benchmarks would be more appropriate, letting users weigh various resources and design scheduling algorithms to leverage them best. Furthermore, sky environments would be greatly enhanced by the ability to negotiate specific latency and bandwidth between various resources provisioned in the clouds. These capabilities are currently unavailable, so we don't explore them in this article.

Finally, the example shown here demonstrates how, using the abstractions we propose, we can layer the platform-as-a-service cloud computing paradigm — in our case, the MapReduce framework — on top of IaaS cloud computing to provision clusters of arbitrary size spanning different providers. It also illustrates the relationship between these two concepts. Our future work will address evolving such clusters dynamically, driven by need as well as availability, and combining the SLA information from various providers to provide differentiated service levels on different infrastructure layers.

Acknowledgments

The US National Science Foundation partially supports this work under grant numbers CNS-0821622, CSR-527448, and OCI-0721867. We also acknowledge the support of the BellSouth Foundation and Shared University Research grants from IBM. We are grateful to the Purdue Rosen Center for Advanced Computing for making its Science Clouds resource available for this research.

References

1. *Open Virtual Machine Format Specification (OVF)*, Distributed Management Task Force, DSP0243, 2009; http://xml.coverpages.org/ni2009-03-23-a.html.
2. C. Sapuntzakis et al., "Virtual Appliances for Deploying and Maintaining Software," *Proc. 17th Large Installation Systems Administration Conf.*, Usenix Assoc., 2003, pp. 181–194.
3. T. Freeman et al., "Division of Labor: Tools for Growth and Scalability of the Grids," *Proc. 4th Int'l Conf. Service-Oriented Computing*, Springer, 2006, pp. 40–51.
4. A. Andrieux et al., *Web Services Agreement Specification (WS-Agreement)*, Open Grid Forum, 2007; www.ogf.org/documents/GFD.107.pdf.
5. M. Tsugawa and J. Fortes, "Characterizing User-Level Network Virtualization: Performance, Overheads and Limits," *Proc. 4th IEEE Int'l Conf. eScience*, IEEE CS Press, 2008, pp. 206–213.
6. M. Tsugawa and J. Fortes, "A Virtual Network (ViNe) Architecture for Grid Computing," *Proc. 20th Int'l Parallel and Distributed Processing Symp.*, IEEE CS Press, 2006.

7. K. Keahey and T. Freeman, "Contextualization: Providing One-Click Virtual Clusters," *Proc. 4th IEEE Int'l Conf. eScience*, 2008, IEEE CS Press, 2008, pp. 301–308.

Katarzyna Keahey is a scientist in the Distributed Systems Lab at Argonne National Laboratory and a fellow at the Computation Institute at the University of Chicago. Her research interests focus on virtualization, policy-driven resource management, and the design and development of cloud computing infrastructure and tools. Keahey created and leads the open source Nimbus project, which provides an infrastructure-as-a-service cloud computing platform as well as other virtualization tools supporting a science-driven cloud ecosystem. Contact her at keahey@mcs.anl.gov.

Maurício Tsugawa is a PhD candidate at the University of Florida. His research interests include computer networks, distributed computing, computer architecture, and virtualization technologies. Contact him at tsugawa@ufl.edu.

Andréa Matsunaga is a PhD candidate at the University of Florida. Her research interests are in the areas of distributed computing, information management, virtualization, and resource management. Contact her at ammatsun@ufl.edu.

José A.B. Fortes is a professor and BellSouth Eminent Scholar at the University of Florida. His research interests are in the areas of distributed computing, autonomic computing, computer architecture, parallel processing, and fault-tolerant computing. He's the director of the US National Science Foundation Industry-University Cooperative Research Center for Autonomic Computing. Contact him at fortes@ufl.edu.