Virtual Distributed Environments in a Shared Infrastructure

A middleware system that integrates and elevates virtual machine and virtual network technologies facilitates the creation of virtual distributed environments in a shared infrastructure.

Paul Ruth
Xuxian Jiang
Dongyan Xu
Sebastien Goasguen
Purdue University

Spanning multiple domains, the Grid provides for the federation, allocation, and management of heterogeneous networked resources and makes them available to a large number of users. Advances in Grid computing technology have contributed to the formation of a wide-area shared cyber infrastructure that provides opportunities for a broad spectrum of distributed and parallel computing applications to take advantage of the massive aggregate computational power available across the Internet. However, realizing the full potential of such a shared infrastructure presents significant research challenges.

Many Grid users are familiar with the traditional job submission and execution model as well as the service-oriented access model as defined by the Open Grid Services Architecture (OGSA).[1] Powered by key enabling technologies such as Globus' Grid Resource Allocation and Management (GRAM),[2] the Grid provides a single detail-hiding interface for requesting and using networked resources to execute jobs that users submit. The widely used job and service models define an appropriate paradigm for resource sharing and program execution.

However, some applications are operational rather than functional, making it difficult to map them to independent jobs or services. Examples of these applications include the fine-grained emulation of real-world systems such as airport operations and antiterrorism exercises, each of which involves numerous dynamic and diverse objects, contexts, and object-interaction patterns.

Furthermore, some applications require specially configured and customized execution environments, including operating systems, network-level services, and application-level services, packages, and libraries. For example, many scientific applications require mathematical and communication libraries such as basic linear algebra subprograms and a message-passing interface (MPI), and a Java application might insist on a specific version of Java virtual machine. It might not always be possible to satisfy such requirements in a shared infrastructure. Moreover, requirements could conflict with each other—for example, the applications might require different versions of the same library.

Finally, it isn't possible to prevent users from running potentially untrustworthy or malfunctioning applications. Developers can introduce bugs or vulnerabilities into an application either inadvertently or deliberately. Well-known applications such as SETI@Home (www.securityfocus.com/bid/7292), for example, have exhibited security vulnerabilities that could be exploited to launch a malicious attack against any machine on the Internet.

As a result, containing the impact of a network attack on an application is critical to prevent any cascading effect on other applications or the underlying shared infrastructure.

We propose a paradigm to accommodate distributed applications that are hard to map to jobs or service instances, require customized execution and network environments, or require strong containment of security risk and impact.

Virtualization introduces a level of indirection between applications and the shared infrastructure.

We've developed a middleware system that integrates and extends virtual machine and virtual network technologies to support mutually isolated virtual distributed environments in shared infrastructures like the Grid and the PlanetLab overlay infrastructure (www.planet-lab.org).[3]

VIRTUALIZATION MIDDLEWARE

Clearly, a shared infrastructure needs to enable mutually isolated distributed environments to complement the job and service-oriented sharing model. Mutually isolated distributed environments should support

- on-demand creation,
- customizability and configurability,
- binary compatibility for applications, and
- containment of negative impact of malicious or malfunctioning applications.

Virtualization is emerging as a promising solution. It introduces a level of indirection between applications and the shared infrastructure. Developers have introduced technologies for the virtualization of machines and networks.

Examples of virtual machine technologies include VMware (www.vmware.com), User-Mode Linux (UML, http://user-mode-linux.sourceforge.net), and Xen.[4] Although their implementations differ, each of these systems offers virtual machines that achieve binary compatibility and networking capability like real machines. Examples of virtual network technologies include VNET[5] and Virtual Internetworking on Overlay Infrastructures (Violin),[6] which both create virtual IP networks for virtual machine communications.

Renato Figueiredo and coauthors first proposed applying virtual machine technology to Grid computing.[7] They identified major advantages of virtual machines for the Grid, including security, isolation, customization, legacy support, administrator privileges, resource control, and site independence. Their In-VIGO[8] and Virtuoso[9] projects are among the first to address Grid resource virtualization and heterogeneity masking. With In-VIGO's VMPlants architecture,[10] the system can automatically create customized virtual machines in a shared execution environment that adhere to the traditional job submission and execution model.

We've taken virtualization to the next level by developing middleware that creates virtual distributed environments in a shared infrastructure such as the Grid and PlanetLab. Our goal is to provide customized and consistent runtime and networking environments for a wide spectrum of distributed applications.

FROM VIRTUAL MACHINES TO VIRTUAL DISTRIBUTED ENVIRONMENTS

In previous work,[11] we extended virtual machine technologies to create virtual private servers. These servers are UML-enabled virtual machines[3] requested and instantiated on demand, each with a regular IP address and a customized installation of OS and application services. Enhancing the underlying host OS guarantees each virtual machine a portion of the physical host's CPU, memory, and bandwidth.

However, virtual machine technologies only achieve isolation between individual virtual machines. A set of virtual machines does not create an isolated virtual network because the machines are addressable to and from other Internet hosts. Moreover, creating multiple virtual networks is difficult because all the virtual machines share the same IP address space, and conflict can occur between the virtual networks.

Virtual network technologies such as Violin and VNET solve this problem by creating high-order virtual IP networks that offer the following features:

- Each virtual network has its own IP address space. The address spaces of different virtual networks can safely overlap.
- Virtual machines in a virtual network aren't visible on the Internet, preventing attacks both from the virtual machines to the Internet and from the Internet directly to the virtual machines.
- Within a virtual network, users can specify and enforce selected (rather than complete) connectivity between virtual machines based on the application's communication pattern.
- A user-specified threshold bounds the traffic volume on each virtual machine interconnection, preventing ill-behaving virtual machines from generating excessive traffic. Furthermore, it isn't possible to tamper with enforcement of the virtual network topology and traffic volume from inside the virtual machines.

Violin achieves network virtualization by introducing user-level communication indirection between the virtual machines and the underlying infrastructure. Virtual network-enabling entities—Violin daemons—are located at this communication-indirection level. These daemons form a user-level overlay network that serves as the underlying virtual network traffic carrier.

Inside the virtual network, the virtual machines use standard IP services to communicate with each other. Below the virtual network, the Violin daemons emulate these services via application-level mechanisms such as User Datagram Protocol (UDP) tunneling. An important benefit of this design is that the daemons can emulate network services that are not widely deployed in the Internet (for example, IP multicast) and enable them in a virtual network.

In VNET, because network virtualization is not completely realized at the user level, host kernel-level devices have to be created to tunnel network traffic. An advanced feature of VNET is its dynamic topology adaptation capability:[9] The virtual network's topology adapts to the virtual machine communication pattern that the application running in the virtual network exhibits, thus dynamically improving the application's runtime communication performance.

Integrating virtual network and on-demand virtual machine creation and customization[11] technologies makes virtual distributed environments a reality. The Violin-based middleware system integrates and enhances such technologies to create virtual distributed environments. For simplicity, we call these environments Violins when there's no ambiguity.

Violins offer the four desirable features of a virtual distributed environment:

- on-demand creation of both virtual machines and the virtual IP network connecting them;
- customization of the virtual network topology and services, OS services, and application services, packages, and libraries;
- achieving binary compatibility by creating the same runtime and network environment under which an application was originally developed;
- containment of negative impact by isolating virtual network address space, limiting both virtual machine resources and inter-virtual-machine traffic rates, and granting users virtual administrator privileges valid only within Violins.

Figure 1 shows a multilayer overview of two Violins on top of a shared infrastructure. In the bottom layer, the physical infrastructure, with heterogeneous networked resources, spans multiple network domains. In the middle layer, middleware systems federate the networked resources to form a shared infrastructure. We deployed our virtualization middleware at this layer. The top layer consists of two mutually isolated Violins, each with its own network, OS, and application services customized for the application running in it.

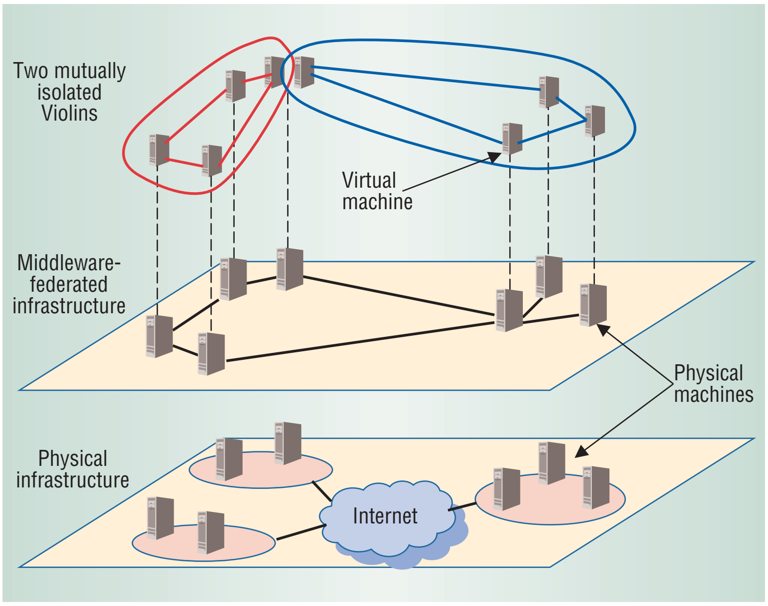
Two mutually isolated Violins
Virtual machine
Middleware-federated infrastructure
Physical machines
Physical infrastructure
Internet

Figure 1. Multilayer overview of two virtual distributed environments (Violins) on top of a shared infrastructure.

VIOLIN DETAILS

Violin realizes the functionalities that connect virtual machines residing on different host machines, allowing communication across a virtual IP network without having a presence in the underlying Internet. Although it's possible to implement Violin's features by extending other virtualization architectures, our implementation takes advantage of UML's user-level execution feature and open source code.

Standard UML virtual machines are instantiated as processes in a Linux user's execution space. Communication outside the host machines occurs through a virtual network interface, or *tap device*, residing in the host's kernel. The UML virtual machine contains a virtual network interface that connects to the host's tap device, and the host acts as a router, forwarding packets between the virtual machine and the physical network. A user will need root-level privileges to safely create the tap device and manage the routing table. In addition, the virtual machine needs an IP address that is routable on the physical network to access the network through the tap device.

Violin bypasses the need for tap devices and lets virtual machines exist in an orthogonal IP address space that is totally decoupled from the Internet. A Violin-enabled virtual machine contains a virtual network interface that does not connect to the host; instead, it connects to a virtual switch (a Violin daemon) running in the host machine's user space. The virtual switch behaves like a physical switch,

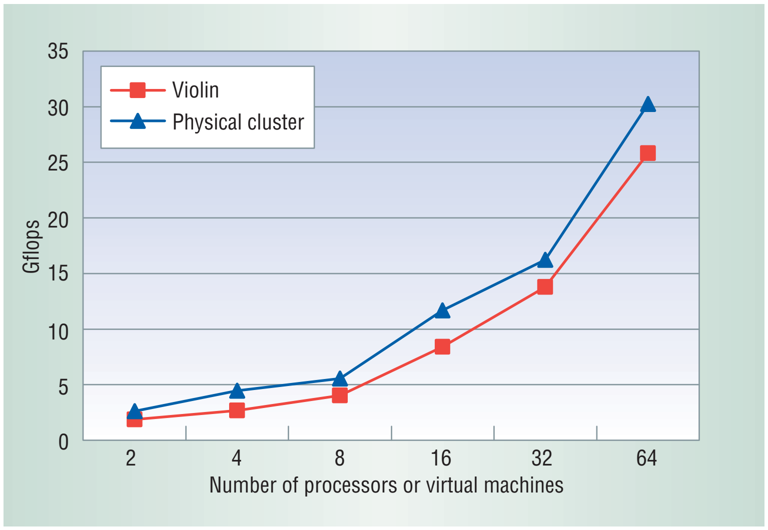

Figure 2. Performance comparison. Comparison of results achieved running High-Performance Linpack (HPL) in a physical cluster and in Violin on top of the same physical cluster nodes shows a 20 percent performance penalty, a scaling trend that shows promise for using Violins in even larger infrastructures.

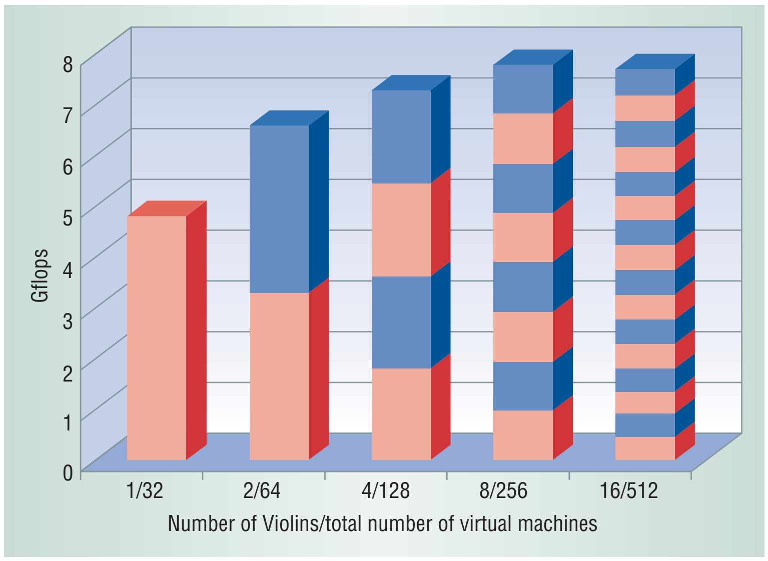

Figure 3. HPL performance with multiple Violins. Each bar represents the aggregate performance observed during each test, with each bar segment representing one Violin's performance. Each Violin consists of 32 virtual machines; thus, in the 16-Violin test, 512 virtual machines concurrently run in 32 cluster nodes.

accepting and forwarding virtual layer-2 frames between virtual machines.

For efficiency, the virtual switches mimic physical network links by tunneling the network frames via the lightweight but unreliable UDP. Using transport protocols (such as TCP) inside the virtual domain can achieve end-to-end reliability between virtual machines. By tunneling virtual network traffic through user-level connections instead of routing the traffic directly through the underlying network, Violin creates a virtual network of virtual machines with only user-level privileges.

EXPERIMENTAL RESULTS

We have successfully deployed the Violin-based middleware system for virtual distributed environments in PlanetLab and in local cluster platforms.

Earlier work demonstrated Violin's communication performance using PlanetLab-based measurements.[6] In these studies, we measured both TCP throughput and Internet Control Message Protocol latency between a pair of virtual machines in a Violin and between the underlying PlanetLab hosts. Our results showed that the virtual machines' performance is constantly within a small degradation factor (less than 15 percent) of the PlanetLab hosts' performance.

HPL performance comparison

To compare Violin's performance with the physical cluster's performance, we conducted experiments using the High-Performance Linpack (HPL) benchmark. Our experiments' methodology was to stress Violin to its limit and find the maximum number of floating-point operations per second (flops) it can achieve relative to the flops achievable in the physical cluster without virtualization.

Each host in the cluster consisted of dual 1.2-GHz Athlon processors and 1-Gbyte RAM running Debian Linux 3.0. A 100-megabyte-per-second Ethernet switch connected the hosts.

We ran HPL in an increasing number of processors in the physical cluster and ran the same increasing number of virtual machines in a Violin on top of the same cluster. For a given number of processors or virtual machines, we tuned HPL parameters including problem size, block size, and process grid shape to reach its maximum performance level.

We set the HPL problem-size parameter as the maximum that Violin can handle and used the same problem size for both the virtual and physical cases. We separately tuned the other parameters for maximum performance in each case. Because a cluster node has dual processors, we created two MPI processes and two virtual machines in each cluster node in the physical and virtual cases, respectively.

Figure 2 shows the experimental results for running HPL in up to 64 processors or virtual machines. These results indicate that Violin constantly achieves approximately 80 percent of the physical cluster's performance. This trend scales to at least 64 virtual machines. Such a performance penalty is small enough to justify Violin's practicality, and the scaling trend shows promise for its use in larger infrastructures.

Sharing among multiple Violins

We also evaluated the performance of multiple Violins sharing the same set of physical nodes. In these studies, we used 32 cluster nodes to create multiple Violins, each with 32 virtual machines. We increased the number of Violins (from 1 to 16) sharing the physical nodes and measured each Violin's performance. In the 16-Violin test, each cluster node supported 16 virtual machines, limiting each to 32 Mbytes of memory.

To make the comparison fair, we found the largest HPL problem size that would run with 32 virtual machines each limited to 32 Mbytes of memory and used the same problem size for all tests in this experiment. We tuned other HPL parameters to find each test's highest performance. Our virtual network technology lets all running Violins use the same set of IP addresses for their virtual machines without any conflict.

As Figure 3 shows, as more virtual machines share the same cluster resources, they each receive a decreasing but equal share of the resources, with little degradation in their aggregate performance. In fact, the aggregate performance increased for up to eight Violins and slightly decreased for the 16-Violin case. Thus, increasing the number of Violins incurs very small overhead in terms of loss of aggregate performance.

Violin communication-pattern profiling

Violin provides a convenient facility for monitoring and profiling the communication pattern and traffic rate among virtual machines. Violin's all-software user-level implementation allows easy extension for network monitoring capability.

Figure 4 shows the communication profile generated by running HPL in an eight-virtual machine Violin. This communication pattern is more irregular and complicated than the classic master-workers pattern seen in distributed applications such as Condor[12] and SETI@home. This suggests that the placement of virtual machines will significantly impact an application's communication performance when the underlying infrastructure involves heterogeneous network connections between physical hosts.

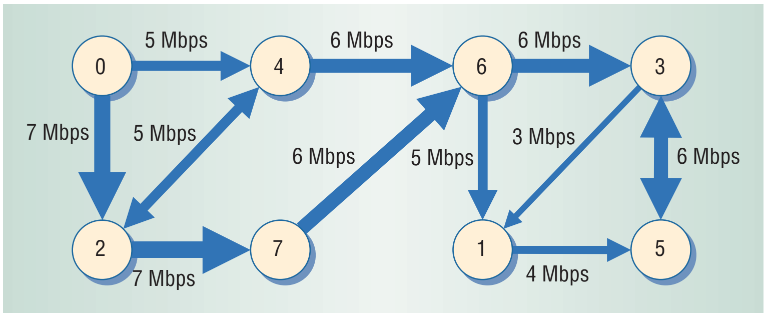

Figure 4. Communication profile. Running HPL in a Violin with eight virtual machines generates an irregular communication pattern. The arrows' direction and thickness indicate the traffic flows and rates between virtual machines.

ONGOING WORK

To make virtual distributed environments more self-adaptive, we're extending our middleware system by adding intelligence to Violins. Each virtual environment will behave like an active organism that manages itself with minimum user attention. Furthermore, a virtual environment will adapt itself at runtime by adjusting its resource allocation, scale, and topology driven by the dynamics of the application running inside. Virtualization technologies offer great flexibility in attaining these goals.

Figure 5 presents a simplified example illustrating support for highly dynamic applications in a next-generation self-adaptive virtual distributed environment. In this example, the application is an agent-based emulation of crowds of people. Each

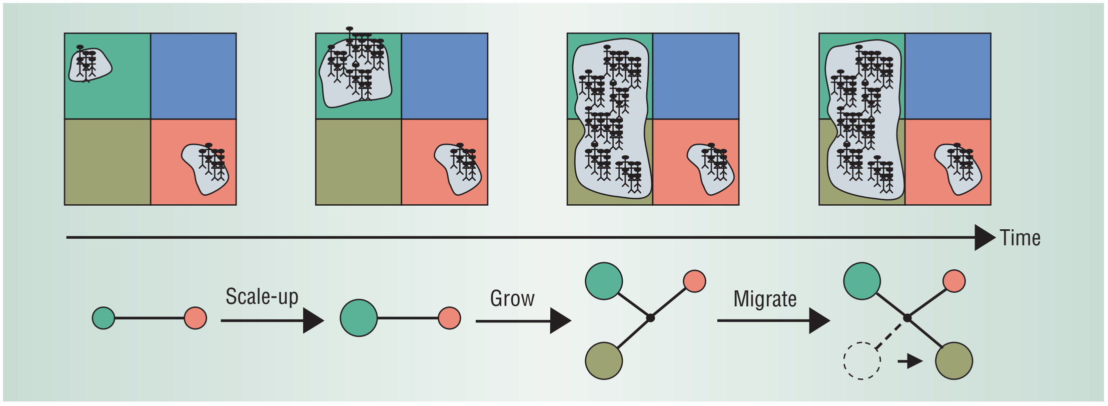

Figure 5. Next-generation virtual distributed environment. The application is an agent-based emulation of crowds of people that studies their mobility and interactions in a specific context. The virtual distributed environment adapts itself to the application's dynamics.

colored square represents a geographic region with unique environmental characteristics. The emulation's goal is to study the mobility of individuals as well as their interactions with each other and with the environment in a specific context such as shopping or commuting.

At the beginning of the emulation (the left-most instance), the few people present are in two of the four regions. At this point, the virtual environment consists of two virtual machines emulating the two regions and the people inside. As time elapses, the crowd in the green region grows, causing a scale-up in the capacity of the virtual machine emulating the green region. Later, more people arrive, and there are people in the brown region. At this point, the virtual environment grows by creating a new virtual machine for the brown region. The last (right-most) instance doesn't involve a change in the emulated world, but instead shows a virtual machine's migration due to a change in the resource availability of the underlying physical host. The virtual environment initiates and performs all the adaptations.

The concept of a virtual distributed environment provides a powerful abstraction for the isolation and customization of runtime and networking systems for different applications running in a shared infrastructure. We propose the integration and elevation of virtualization technologies to support mutually isolated virtual distributed environments in shared infrastructures, such as the Grid and PlanetLab.

Our experimental results using the Violin-based virtualization middleware support further research and wider deployment of virtual distributed environments as a new paradigm for distributed computing. In particular, next-generation Violins will possess the intelligence for self-provisioning and adaptation, providing more agile and accommodating virtual distributed environments for highly dynamic distributed applications. ■

Acknowledgment

The US National Science Foundation partly funded this research under grants SCI-0438246 and SCI-0504261.

References

1. I. Foster et al., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Open Grid Service Infrastructure WG, Global Grid Forum, June 2002; http://www.globus.org/research/papers/ogsa.pdf.
2. I. Foster and C. Kesselman, "Globus: A Toolkit-Based Grid Architecture," *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, pp. 259-278.
3. L. Peterson et al., "A Blueprint for Introducing Disruptive Technology into the Internet," *Proc. ACM Workshop on Hot Topics in Networking* (HotNets-I), ACM Press,2003, pp. 59-64.
4. P. Barham et al., "Xen and the Art of Virtualization," *Proc. ACM Symp. Operating Systems Principles* (SOSP), ACM Press, 2003, pp. 164-177.
5. A. Sundararaj and P. Dinda, "Toward Virtual Networks for Virtual Machine Grid Computing," *Proc. 3rd Usenix Virtual Machine Technology Symp.*, Usenix, 2004, pp. 177-190.
6. X. Jiang and D. Xu, "Violin: Virtual Internetworking on Overlay Infrastructure," *Proc. 2nd Int'l Symp. Parallel and Distributed Processing and Applications*, LNCS 3358, Springer-Verlag, 2004, pp. 937-946.
7. R. Figueiredo, P. Dinda, and J. Fortes, "A Case for Grid Computing on Virtual Machines," *Proc. 23rd Int'l Conf. Distributed Computing Systems* (ICDCS), IEEE CS Press, 2003, pp. 550-559.
8. S. Adaballa et al., "From Virtualized Resources to Virtualized Computing Grid: The In-VIGO System," *J. Future-Generation Computing System*, to appear.
9. A. Sundararaj, A. Gupta, and P. Dinda, "Dynamic Topology Adaptation of Virtual Networks of Virtual Machines," *Proc. 7th Workshop Languages, Compilers, and Runtime Support for Scalable Systems*, 2004, http://www.tlc2.uh.edu/lcr2004/Final_Proceedings/Sundararaj.pdf.
10. I. Krsul et al., "VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing," *Proc. IEEE/ACM Supercomputing*, IEEE CS Press, 2004, p. 7.
11. X. Jiang and D. Xu, "SODA: A Service-on-Demand Architecture for Application Service Hosting Utility Platforms," *Proc. 12th IEEE Int'l Symp. High-Performance Distributed Computing* (HPDC-12), IEEE CS Press, 2003, pp. 174-183.
12. D.H.J Epema et al., "A Worldwide Flock of Condors: Load-Sharing among Workstation Clusters," *J. Future-Generation Computer Systems*, vol. 12, no. 1, 1996, pp. 53-65.

***Paul Ruth** is a PhD candidate in the Department of Computer Science at Purdue University and a member of the Laboratory for Research in Emerging Network and Distributed Services (Friends). His research interests include virtualization, high-*

performance computing, and Grid computing. Ruth received a BS in computer science and mathematics from Baker University, Kansas. He is a member of the IEEE. Contact him at ruth@cs.purdue.edu.

***Xuxian Jiang** is a PhD candidate in the Department of Computer Science at Purdue University and a member of the Laboratory for Research in Emerging Network and Distributed Services (Friends). His research interest is in virtualization-based system security and distributed computing. Jiang received an MS in computer science from Xi'an Jiaotong University, China. He is a member of the IEEE, the ACM, and Usenix. Contact him at jiangx@cs.purdue.edu.*

***Dongyan Xu** is an assistant professor in the Department of Computer Science at Purdue University and director of the Laboratory for Research in Emerging Network and Distributed Services (Friends). His research interest is in virtualization-based distributed computing and system security. Xu received a PhD in computer science from the University of Illinois at Urbana-Champaign. He is a member of the IEEE, the ACM, and Usenix. Contact him at dxu@cs.purdue.edu.*

***Sebastien Goasguen** is a research scientist and director of the Grid and Distributed Computing Group within the Department of Information Technology at Purdue University. His research interests include applied computational science, parallel programming, virtual machines, and Grid computing. Goasguen received a PhD in electrical engineering from Arizona State University. He is a member of the IEEE. Contact him at sebgoa@purdue.edu.*