# Applying Virtualization and System Management in a Cluster to Implement an Automated Emulation Testbed for Grid Applications*

Rodrigo N. Calheiros, Mauro Storch, Everton Alexandre, César A. F. De Rose
Pontificial Catholic University of Rio Grande do Sul
Porto Alegre, Brazil
{rcalheiros,storch}@inf.pucrs.br,{everton.alexandre,cesar.derose}@pucrs.br

Marcus Breda
HP Brazil R&D
Porto Alegre, Brazil
mbreda@hp.com

## Abstract

*Although grid systems have evolved in such a way that they are largely used both in industry and academy, techniques to test and evaluate them, such as simulation and emulation, have limitations on both their applicability and their reliability. We are investigating the utilization of paravirtualization techniques merged with systems management tools to build an automated emulation framework for grid experiments. This framework accesses standard network resources to manage communication among virtual nodes, allowing virtual machines to behave like a real grid environment. The development of this framework involves the mapping of virtual machines to physical hosts, automatic deployment and management of virtual machines, automatic configuration of virtual network and experiment control. In this paper, we address these issues and present results demonstrating the feasibility and advantages of our approach.*

## 1. Introduction

Grid computing has become an important technology for both academy and industry. Several projects rely on grid to provide processing power to their experiments. Moreover, enterprise grids are becoming an alternative to comply with internal demand of resources of many companies. However, as the interest in grid research and utilization rises, so do the complexity of researched systems.

Intricate issues that arise when running grid experiments concern resource access and replication of experiments because of the lack of control over the elements spread among several administrative domains that interfere in the tests. An alternative approach to grid experimentation is simulation [12, 22], which simplifies the low-level details of the environment in order to represent high-level aspects of the system in a reasonable time, at the cost of loss of accuracy of the experiment results.

Another approach for experimentation of grid systems is emulation [17]. In this approach, real applications run in a testbed which emulates the behavior of real grid infrastructures. The drawback of system emulation is that it requires a complex software stack encompassing emulation, controlling of physical resources, simulation of some system structures, and sometimes modification in the operating system supporting the testbed. Emulation may also require unusual hardware to adequately host the experiments.

These disadvantages of emulation can be avoided if virtualization [20] tools were used to simplify the emulation software. In this approach, the Virtual Machine Monitor (VMM) controls the resources multiplexing, and the emulation software uses VMM services to generate and configure virtual nodes. However, currently available virtualization-based emulation tools require that users configure the virtual environment, either by direct operation of the VMM or by the use of some virtualization-support tool. In both cases, it is required that the user learn how to use these extra tools.

To circumvent this limitation, we present an automated emulation framework for grid computing experiments. To achieve such a goal, two main aspects were addressed (i) application of systems management protocols [11, 21] to automate the installation and execution of experiments, and (ii) solution of the mapping problem of virtual machines

*This work was developed in collaboration with HP Brazil R&D.

1550-6533/08 $25.00 © 2008 IEEE
DOI 10.1109/SBAC-PAD.2008.12

(VMs) to hosts as well as the mapping of virtual links between virtual machines to real paths among hosts. To the best of our knowledge, no emulation tool based in virtualization addresses both these issues.

## 2. Virtualization and Paravirtualization

In this paper, the word *virtualization* is used with the meaning of *system virtualization*. This is the technique that allows a single computer to host one or more virtual machines (VMs). Each VM runs a complete operation system. Each operating system has its own memory area, CPU time, system resources and so on. However, resources used by the virtual machine's operating system are only a subset of the overall resources of the host. Sharing and access control of resources are provided by a software layer between hardware and VMs called Virtual Machine Monitor (VMM) [20].

Operating systems running over a VMM were developed for a given hardware platform and do not have to be modified to run in a virtual environment. However, to increase system performance, it is possible to apply the technique called paravirtualization, where the operating systems running over the VMM are adapted to become aware of the virtual environment. Recent hardware-assisted virtualization [14] allows unmodified operating systems to run over paravirtualization software.

The VMM used in our prototype is Xen [3], a paravirtualization software. In Xen terminology, VMs are known as *domains*. One of such domains – called dom0 – is a privileged domain that manages other domains as well hosts the real device drivers. The other domains – called domU – do not have direct access to hardware. Instead, the device drivers running on these domains only forward requests to the real device drivers in dom0.

When a VM in Xen is created, the amount of memory of the VM must be defined. This value can be dynamically changed via dom0. The CPU usage is controlled with both the assignment of virtual processors to virtual machines and the definition of the length of the CPU time slice to each VM.

Xen network architecture presents a set of real and virtual components. Xen's dom0 is responsible not only for the management of this components but also for the support of network communication among virtual machines. Virtual switches are used in dom0 to connect both real and virtual interfaces. A virtual machine can have one or more virtual interfaces connected by one or more virtual switches. These virtual switches can run in one of three modes: bridge, router, and NAT.

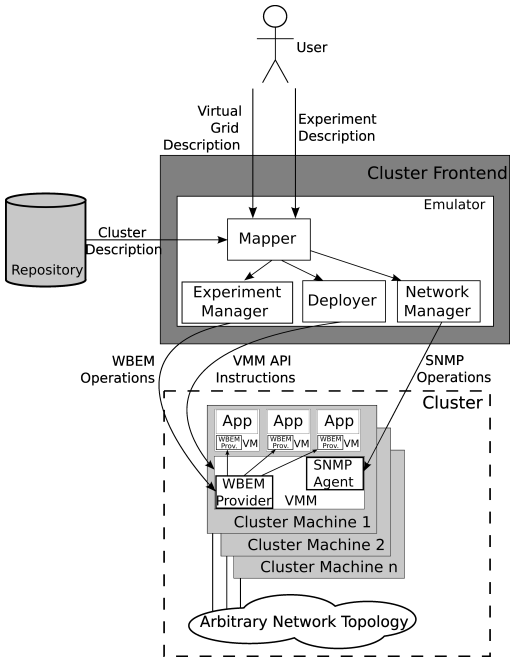


**Figure 1. Emulator architecture.**

## 3. The Automated Emulation Framework

The Automated Emulation Framework architecture is represented in Figure 1. The target architecture of the framework is a cluster of workstations. The emulator runs in the cluster frontend. The cluster nodes can be either homogeneous or heterogeneous, and they can be connected by any network topology. The VMM is mandatory only in the cluster nodes (it is not required that the cluster frontend run a VMM), and all the nodes must run the same version of it. Each module of the framework interacts with one component of the virtualized system. This interaction is explained in Section 3.2.

### 3.1. Installation and Configuration Workflow

The first step to run an experiment in the proposed framework is the installation and configuration of the virtual grid. It is done according to the workflow presented in Figure 2.

The system input is the description of the virtual environment and the experiment, supplied by the user via a graphical user interface. The description of the cluster is known by the emulator and includes information about the network (e.g., cluster network topology), specification of each machine (e.g., their capacity, amount of RAM memory available, and network addresses), version of the virtualization software in use, and amount of physical resources in use by the VMM on each machine. The description of the virtual

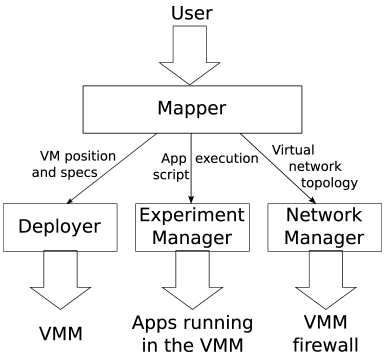


**Figure 2. Installation and configuration workflow.**

system supplied by the user contains the sites, virtual hosts belonging to each site, its configuration, and network characteristics.

A Mapper module, described in the next section, uses the user input to determine both where each VM will run and the physical path that will correspond to each virtual link between virtual nodes. After the specification of the target environment, the Deployer module automatically install the equivalent virtual grid and starts virtual machines on the hosts (i.e., the cluster nodes). Afterward, the Network Manager module configures the network environment to comply with user demands. After these steps the virtual grid is built and the required experiment is started by the Experiment Manager.

## 3.2. Framework Components

This section describes each component of the proposed emulation framework.

### 3.2.1 Mapper

Mapping of virtual resources to real resources is a key issue not only for emulation software, but also for several virtualization projects. The mapping problem, in the context of emulation, is addressed in [19], namely the solver `assign`. However, because our approach uses also virtualization, we must consider variables that the basic problem does not consider: `assign`, for instance, does not consider utilization of memory, CPU, and other physical resources in its assignments.

In [23], the problem of setting virtual routes representing real links between machines is formalized and proved to be an NP-Hard problem. Thus, solutions for this problem must be addressed with the use of heuristics. The requirements of a solver for the mapping problem in the context of our work are the same requirements presented by Ricci *et al.* [19]: (i) the problem must be solved in a short period of time, avoiding that a significant part of the experiments time be spent in this stage, and (ii) the solver must reduce the overall usage of physical resources for a single mapping, allowing multiple executions of experiments at the same time (space sharing).

The current heuristic in use in the Mapper module is a greedy heuristic that tries to reduce communication among hosts, placing guests with links between them in the same host, if possible, or randomly choosing a host otherwise. Development of new heuristics and methods to test and evaluate them are research subjects of this project.

### 3.2.2 Deployer

After mapping the guests to the hosts, it is necessary to start the virtual machines in the selected hosts. The Mapper generates an abstract representation of the virtual environment. Afterward, the Deployer module translates this internal representation to a representation understandable by a VMM deployment tool, which will load and configure each VM in the hosts selected by the Mapper.

This module is composed of two parts: the first one, the Converter, translates the internal representation of the virtual grid to the language of the specific deployer. The second part, the Actor, transmits to the VMM the instructions generated by the Converter. The Actor can be an external deployment tool or can be a module from the emulator.

The Deployer module is the emulator component that provide support to different VMMs. Because it is the only framework module that interacts directly with the VMM, different VMMs could be used if a module translating the framework internal virtual environment representation into the language of the new deployment tool were added to it. In our current implementation, the VMM currently in use is Xen [3], and the deployment is made with the XSM [9] tool.

### 3.2.3 Network Manager

The Network Manager module has two functions. The first one is to provide the grid behavior for the virtual machines taking part in the emulation. To provide such a behavior, the module must provide isolation among VMs that virtually belongs to different networks. This module must also generate the network conditions required by the user. Typically, it means to set the bandwidth and the latency between guests in such a way that the communication behaves like a wide area network.

The second function of this module is to offer "virtual services" to the virtual environment. These services are confined DHCP and DNS servers that run as threads of the emulator and avoid the use of real DHCP and DNS servers to attend virtual nodes.

The automatic configuration of the environment is made with the Simple Network Management Protocol (SNMP) [21]. This protocol was chosen because it is a fast and simple system management protocol, even though powerful enough to support the low level configuration required by the module.

The Network Manager Module implements the SNMP manager, while the SNMP agents run in the VMM on each cluster node. The agents receive SNMP operations from the manager and perform the received configuration in their hosts. The possible configuration include applying network isolation and setting of network link parameters such as bandwidth and latency.

In our prototype we use Xen, where the configuration is applied with `iptables` configuration rules. It is done in the `dom0`, a special domain that has high privileges accessing the system.

To use other VMM, only the SNMP agent must be ported to it. Porting the SNMP agent means translating each configuration operation executed by the Xen agent to the equivalent operation in the target VMM.

#### 3.2.4 Experiment Manager

The Experiment Manager controls the execution of the experiment, according to the description supplied by the user. This module must address issues such as how the applications will be configured, how it will be started and stopped, and how the results of the execution will be returned to the users.

Regarding application configuration, this issue is trivial in a virtualization-based environment: both the application and their configuration must be present in the operating system image being loaded in the virtual machine. This way, loading an experiment means loading a set of pre-built virtual machine images stored in a place accessible by the emulation system.

The other issues, start and stop of applications and retrieval of results, are currently being addressed with the use of the WBEM system management protocol [11]. WBEM was chosen to be used in this module because, unlike the SNMP, it is able to control high-level applications, in spite of being slower and more complex than SNMP.

The Experiment Manager implements the WBEM client, while the WBEM provider is present in the virtual machine image. Thus, replacement of VMM does not require changes neither in the Experiment Manager Module nor in the WBEM provider. It is required, however, an implementation of the WBEM provider for each operating system running in the VMs.

## 4. Testing Grid Applications

Testing grid applications is not an easy task and involves many variables that directly affect the application. One of the main points to guarantee a good quality assurance testing [16] is being able to replicate and reuse the same running environment. Our emulator allows it, using virtualization to build the grid environment.

Testing can be mapped in a matrix with levels and techniques. Four different testing levels are presented in [16]: component, integration, system, and acceptance. We believe that our solution brings a great improvement on system-level tests, being an interesting platform for scalability and fault tolerance tests.

Scalability test may be very limited using only physical systems due to limited hardware resources present in a testing environment. A virtualized environment allows measurements regarding the system behavior with more nodes than the available in the physical environment.

Another relevant achievement in our solution is the possibility to apply fault tolerance testing, especially when it is necessary to simulate a fail of a node, or when it is required to introduce network interference (such as huge latencies) or even when the intention is to verify the behavior of a system with slow nodes.

Performance testing is not easily applied in virtualized environments because it adds an extra layer that degrades system performance.

## 5. Evaluation

In this section, we present a series of tests that show the feasibility and advantages of our approach for grid emulation. The cluster hosting the virtual grid is composed of 8 Pentium 4 2.8GHz with 1MB of cache and 2560MB of RAM memory. Cluster machines are connected by a dedicated Fast Ethernet switch. Machines run Xen VMM 3.1, and the Xen's dom0 uses 328MB of the available RAM memory. Thus, 2232MB were available to the guests on each host. No network traffic but the one generated by this experiment was present in the physical environment during the tests.

### 5.1. Evaluating Network Emulation

Our first test had the goal of evaluating the effectiveness of our network bandwidth control. In this test we considered both communication between VMs hosted in a same host (referred as *local*) and communication between VMs hosted in different machines (referred as *remote*).

To evaluate bandwidth emulation, a 300MB file was transmitted in each case (local and remote VMs), and the bandwidth between sender and receiver varied on each test.

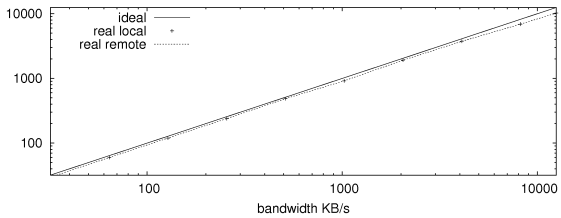


**Figure 3. Network bandwidth between local and remote VMs.**

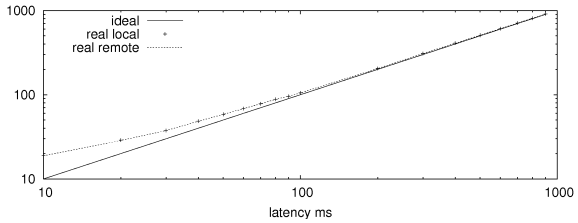


**Figure 4. Network latency between local and remote VMs.**

Secure Copy Protocol was used to transmit the file among the nodes. Data obtained in this test were used to compute the average latency.

Figure 3 shows the result of this test. In this figure, horizontal axis represents the demanded value of bandwidth, and the vertical axis represents the observed bandwidth in the measurements. Each point in the figure represents an average of 100 samples, with a standard deviation of 2%. The straight line represents the configured value (this is the expected value if the configurator had an accuracy of 100%) for the bandwidth. The dotted lines represent the values of bandwidth obtained in each of the test scenarios (local and remote communication).

The deviation from the configured value to the measured value increases as the desired bandwidth increases, both in local and remote communication. This effect is caused by the bandwidth control mechanism, whose operation overhead increases with the increase of the configured bandwidth value. Thus, for low bandwidth values the overhead are negligible, while for high bandwidth values it has a higher effect in the system. In a distributed environment, low values of bandwidth are more common than high values, so in the most experiments this approach will generate a low deviation in the obtained bandwidth.

To evaluate latency emulation, ICMP echo/request packages were transferred between both local and remote VMs. Each test sent and received 1000 ICMP packages, and the average value of the 1000 values composes one sample. From this data the average latency was calculated and presented in Figure 4.

Figure 4 shows the results of the test. In this figure, horizontal axis represents the demanded value of latency, and the vertical axis represents the observed latency in the measurements. The straight line represents the configured value for the latency. The dotted lines represent the values of latency obtained in each of the test scenarios (local and remote communication). Each point in the figure represents an average of 32 samples, with a standard deviation smaller than 1%. It is noticeable that the obtained latency values have a higher deviation from the configured value to lower latency values.

The deviation decreases as the latency value increases. This effect is caused by the latency control mechanism, that causes an overhead almost constant, independent of the configured value of the latency. So, the impact of this overhead is greater for low values of latency than for higher values. In distributed environments, high values of latency are more common than low values, so in most cases this approach will also generate low deviation.

## 5.2. Running a Grid Experiment

The next test aimed at show the capacity of our emulation framework to successfully run a grid application. The grid middleware used was the OurGrid [8]. In the OurGrid system, each peer is responsible for communicating with other peers in order to get resources to comply with its local demands and also to provide resources to other peers, in an economic model known as *network of favors* [1]. Grid resources, which run the software named UserAgent or the one named SWAN, are supplied to grid users, through the MyGrid scheduler.

The virtual environment built in the test is composed of three sites, each one containing a proxy which is connected to the other proxies, as shown in Figure 5. Each site has a different number of hosts. The bandwidth between each site is 1Mbps. The total number of virtual machines was 32, each one with 256MB of RAM memory. The machines labeled `vm00` on each virtual site are the OurGrid peers, and they are responsible for communicating with other peers in order to get machines to meet their users' demands. Peers discover each other through the `corepeer` component, which run in site 1, in the same VM running the OurGrid's `peer`. Each VM belonging to a site has access to the other VMs belonging to the same site, emulating an Ethernet LAN.

One machine in each site is used to run the application. These machines run the `MyGrid` component of the OurGrid, accessing the grid through the peer in the same site where the user is located. The rest of the machines in the sites are worker machines (`GuMs` in OurGrid terminology) and run the `UserAgent` component of OurGrid,

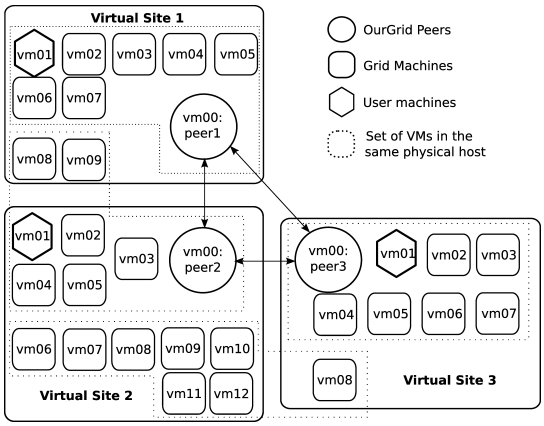


**Figure 5. Virtual grid environment built for the tests.**

being able this way to execute grid tasks. No other services besides the OurGrid components were running in the VMs used in the experiment during these tests.

Three users, each one located in a virtual site, run an application. The application is composed of 20 tasks. Each task stages in 1 MB of data, sleeps during 10 minutes and stages out 1MB of data. In order to provide more data transfer among sites, applications were configured to run in a site different from the one they were triggered. So, for example, tasks from the user 1, located at site 1, can only run on sites 2 and 3. All the applications were triggered at the same time.

The mapping generated by the mapper is represented by the dotted lines around the nodes in Figure 5. As expected, the mapper was able to place 8 virtual machines in each host. Thus, only 4 from the 8 available physical hosts were used in this test.

In order to validate the environment, the OpManager network manager [15] was used to discover the network topology of the virtual environment. The three sites were correctly identified. Virtual machines belonging to each site were correctly identified as well. No direct access was possible between machines hosted in the same host but belonging to different sites. The three grid jobs successfully executed in the environment. Due to OurGrid scheduling characteristics, some of the resources were preempted to be donated to another user. Even though, the jobs completed after 4 resources preempted from user 1, 3 resources preempted from user 2, and none from user 3. User 1 received 18 resources, user 2 received 10 resources and user 3 received 11 resources. As tasks finished their execution in different moments, some resources were relocated to another user, that explains why the sum of resources used is greater than the amount of resources in the environment.

### 5.3. Modifying Experiment Behavior

We were able to use the same grid environment used in the previous test to perform another grid experiment. In this new experiment, three users executed the same job presented in the last section. However, this time all the users were located in the site 1. Thus, only resources from sites 2 and 3 were used. This test did not require a new installation and configuration process. It was only necessary to run the user application from the new site.

Modification of experiment behavior without building a new environment is a very useful feature of our emulation framework, because it allows not only a quickly observation of the effects of different conditions in the grid application, but also a quickly reuse of the environment. As stated in Section 4, replication and reuse of running environments are key points to good quality assurance tests. So, this experiment shown that our framework allows users to have effective quality assurance tests.

### 5.4. Scaling the Virtual Grid

Next, we tested the scalability of the environment. We successfully scaled the grid from 32 to 128 machines, using all the cluster nodes and reducing the amount of memory of each VM from 256MB to 128MB. To run this new experiment, the previous experiment was removed from the cluster and a new installation and configuration process were run. The new machines were equally distributed between site 2 and site 3. The three users were in the same site, as in the previous experiment.

It shows that our emulation framework effectively supports scalability test. As pointed in Section 4, these tests are limited in physical systems. As our proposal is able to circumvent this restriction of physical systems, it emerges as an interesting alternative for performing scalability tests.

### 5.5. Comparing to a real-world experiment

To compare results of an emulated experiment to a real-world experiment, we emulated the measurement of makespan of jobs running in a OurGrid grid using the SRS scheduler [4]. The environment used for the real world test is composed of 50 machines in 2 OurGrid sites, each one located 4000 Km apart. One site has been used as a resource consumer, and the other has been used as a resource provider, which hosted a cluster whose machines have been opportunistically delivered to the grid. The supplier had 48 grid machines plus one OurGrid peer. The consumer site had only one machine, which contained both the peer and the MyGrid scheduler.

The grid job executed in both experiments contains 12 tasks, each one sending a file, executing a sleep call of 5

**Table 1. Observed makespan of jobs running in a real environment and in the emulated environment.**

| File size | Real | Emulated | Deviation |
|---|---|---|---|
| 0 | 313s | 309s | 1.28% |
| 100kB | 398s | 387s | 2.76% |
| 1MB | 1283s | 1227s | 4.36% |
| 10MB | 10100s | 9138s | 9.52% |

minutes, and receiving a file of the same size of the file sent. The job was executed 4 times: the first one without file transferring and the others with different file sizes: 100KB, 1MB, and 10MB. To simulate the dynamism of a grid environment, resources are randomly removed from the grid every 10 minutes.

The network parameters used in the virtual network were obtained with the observation of the bandwidth obtained in a data transfer using scp (for the bandwidth) and with the latency measured by the hping2 tool, which were respectively 2Mbps and 200ms.

Table 1 presents the observed makespan of the job in both real and virtual environment. It is also shown the deviation, e.g., the percentage's difference between the result observed in the real environment and the result from the emulated environment. The deviation between the real and emulated results were less than 10% for all the cases. However, this value increases with the size of the file being transferred. It happens due to the cumulative error in the network emulation: when small files are transferred, the network is less demanded, and the difference between the real and the virtual network becomes smaller. However, the emulated network is "faster" than the real network. So, when larger files are transferred, the difference between the emulated and the real network causes a bigger influence in the results of the experiment.

There are several causes of the deviation in the emulated and real networks. Some of them are the following:

**Error in the acquisition of network parameters:** Measurement of network parameters (latency and bandwidth) is a difficult task, specially when it involves machines belonging to different administrative domains, in which common methods to evaluate it (e.g., ICMP echo request to measure latency) are blocked by systems administrators. To circumvent it, we used tools running in the application layer. So, it is expected an inaccuracy in the values used to set the emulated environment. It is possible to reduce the error using more accurate methods to acquire bandwidth and latency values.

**Fluctuation in the real network traffic:** The network parameters are not static, and variation in the network traffic led to variation in the available latency and bandwidth. So, the values of latency and bandwidth we measured had varied during the execution of the real environment. In the emulated environment, on the other hand, these parameters were statically defined. In spite of being possible to apply techniques to insert variation in the experiment, it was not made in this early evaluation. This fluctuation can be injected in the experiment in the following way. The user supplies the demanded value of latency and bandwidth in the form of a probability distribution. Then, periodically the Network Manager reconfigures network links with a value randomly selected according to the probability distribution. This way, it is possible to decrease the deviation between real and emulated experiments.

**Error in the network emulation:** Finally, there are also differences between the configured network parameters and the values obtained in the emulated network, as shown in Section 5.1. However, as presented in the same section, in spite of the network mechanism interfere in the emulation, this interference is not critical in the presented evaluation.

Even though all those factors can interfere in the results, we were able to get results that are close to the ones observed in the real experiment. As the results in Table 1 suggest, the error tends to accumulate in time. So, short-time experiments tend to have better results than long-time experiments. Nevertheless, even an experiment that took almost 3 hours to complete in a real environment presented an deviation below 10%, what shows that, in spite of being a prototype, the emulation framework presented in this paper is able to successfully emulate grid systems.

## 6. Related Work

Several researchers have been developed grid and distributed systems emulation testbeds. Emulab [10], is one well-succeeded project in this area. It uses BSD jail in a network-complex environment to multiplex virtual hosts into physical nodes. Our approach, in spite of supporting only a subclass of the experiments supported by Emulab, can run in regular clusters, and requires less complex software to run, as it exploits capabilities of VMM software.

Another recent approach for distributed systems emulation is the NET emulator [13], which allows emulation of distributed and ad-hoc networks through the modification of the operating systems network stack. This kind of approach restricts the diversity of environments being emulated to the ones whose operating system is exactly the same modified one. Approaches using paravirtualization (as the one presented in this paper) are expected to be freed from this limitation as more virtualization-enabled hardware become available in physical environments.

Several projects apply VMM to support emulation of grids and other distributed systems in a distributed environment, such as V-DS [18], V-eM [2], TestGrid [7], and NEP-

TUNE [5]. However, all of them lack in offer either automatic mapping of guests in hosts with arbitrary topology or automatic building and execution of arbitrary experiments.

Both V-DS and NEPTUNE require that users perform deployment and configuration of the guests. The later is able to perform the mapping, while the former requires that users choose the host for every guest. V-eM supports only low level experiments (such as network protocol tests) in switched homogeneous clusters. TestGrid does not support network emulation, and requires that users choose the host for every guest, which can be configured and deployed with the GridBuilder [6] tool.

## 7. Conclusion and Future Work

In this paper we investigate the utilization of virtualization techniques combined with systems management to build an automated emulation framework for grid experiments. This approach allows the configuration of a set of virtual machines in such a way that they behave as a grid infrastructure described by the user, implementing a controlled and scalable environment to test grid software.

The main difference of our work and other emulators based in virtualization technology is the ability to not only automatically create, configure, and deploy the environment but also automatically run the experiment. This is possible with a modularized framework based on resource management standards like SNMP and WBEM.

In spite of being a prototype, the framework presented in this paper was able to successfully emulate a grid infrastructure, allowing the user to run a grid experiment, reproduce the experiment with different user behaviors, and to scale the environment. We believe it is already a very interesting alternative to traditional techniques for testing and evaluation of grid applications.

As future work, we will further investigate the adaptive emulation and the mapping strategies applied in the framework.

## References


[1] N. Andrade, F. Brasileiro, W. Cirne, and M. Mowbray. Automatic grid assembly by promoting collaboration in peer-to-peer grids. *Journal of Parallel and Distributed Computing*, 67(8):957–966, 2007.

[2] G. Apostolopoulos and C. Hassapis. V-eM: A cluster of virtual machines for robust, detailed, and high-performance network emulation. In *14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2006.

[3] P. Barham et al. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 2003.

[4] R. N. Calheiros, T. Ferreto, and C. D. Rose. Scheduling anf management of virtual resources in grid sites: the site resource scheduler. *Parallel Processing Letters*, 2008.

[5] R. Canonico et al. Virtualization techniques in network emulation systems. In *Workshop on Virtualization/Xen in HPC Cluster and Grid Computing Environments*, 2007.

[6] S. Childs, B. Coghlan, and J. McCandless. GridBuilder: A tool for creating virtual grid testbeds. In *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, 2006.

[7] S. Childs et al. A virtual TestGrid, or how to replicate a national grid. In *The 15th IEEE International Symposium on High Performance Distributed Computing (CDROM)*, 2006.

[8] W. Cirne et al. Labs of the world, unite!!! *Journal of Grid Computing*, 4(3):225–246, 2006.

[9] F. Franciosi et al. Deploying and managing Xen sites with XSM. In *Workshop on Virtualization/Xen in HPC Cluster and Grid Computing Environments*, 2007.

[10] M. Hibler et al. Feedback-directed virtualization techniques for scalable network experimentation. Technical Note FTN-2004-02, University of Utah Flux Group, 2004.

[11] C. Hobbs. *A practical approach to WBEM/CIM management*. Auerbach, 2004.

[12] A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: the SimGrid simulation framework. In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2003.

[13] S. Maier, D. Herrscher, and K. Rothermel. Experiences with node virtualization for scalable network emulation. *Computer Communications*, 30(5):943–956, 2007.

[14] G. Neiger et al. Intel virtualization technology: Hardware support for efficient processor virtualization. *Intel Technology Journal*, 10(3):167–177, 2006.

[15] OpManager. OpManager. `http://www.opmanager.com/`, 2008.

[16] J. F. Peters and W. Pedrycz. *Distributed Systems: Concepts and Design*. Wiley, 1999.

[17] B. Quétier and F. Cappello. A survey of grid research tools: simulators, emulators and real life platforms. In *17th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation (CD-ROM)*, 2005.

[18] B. Quétier, M. Jan, and F. Cappello. One step further in large-scale evaluations: the V-DS environment. Research Report RR-6365, INRIA, 2007.

[19] R. Ricci, C. Alfeld, and J. Lepreau. A solver for the network testbed mapping problem. *ACM SIGCOMM Computer Communication Review*, 33(2):65–81, 2003.

[20] J. E. Smith and R. Nair. *Virtual Machines: Versatile platforms for systems and processes*. Morgan Kauffmann, 2005.

[21] W. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison Wesley, 3 edition, 1999.

[22] A. Sulistio, G. Poduval, R. Buyya, and C.-K. Tham. Constructing a grid simulation with differentiated network service using GridSim. In *Proceedings of the 6th International Conference on Internet Computing*, 2005.

[23] A. I. Sundararaj et al. An optimization problem in adaptive virtual environments. *ACM SIGMETRICS Performance Evaluation Review*, 33(2):6–8, 2005.