Virtual Machines Networking for Distributed Systems Emulation

Mauro Storch[1], **Rodrigo N. Calheiros**[2], **César A. F. De Rose**[3]

[1]Instituto de Pesquisas Eldorado
Porto Alegre, Brazil
Email: mauro.storch@eldorado.org.br

[2]Department of Computer Science and Software Engineering
The University of Melbourne, Australia
Email: rnc@unimelb.edu.au

[3]Pontifical Catholic University of Rio Grande do Sul
Porto Alegre, Brazil
Email: cesar.derose@pucrs.br

Abstract. *Distributed systems are the dominant platform for any application, both in academic and industry. However, testing of applications in such a platform is hard, due to the complexity of the elements that compose these systems. Emulation is an alternative way to improve software quality for distributed applications, allowing analysis of the application behavior in a given environment before its deployment. To emulate distributed systems, virtualization can be applied, as it allows creation of a large-scale controlled environment with few physical resources. Such an approach, however, requires application of network management of virtual machines. This paper presents approaches for network virtualization and a description of how it can be used to create a network of virtual machines for emulation of different distributed systems configurations. This work is part of a system designed for creating a virtual environment, based on virtualization technology, to evaluate distributed applications.*

1. Introduction

Virtualization is an outstanding technology which allows rich control of physical resources, enabling full exploitation of machines capacity at a negligible overhead cost. Current virtual machine monitors [Devine et al. 1998, Barham et al. 2003] offer a high degree of control over the sharing of physical resources, such as CPU and memory. It is easy, using such tools, to deploy several isolated virtual machines (VMs) in a single host, possibly running different operating systems.

The next step on application of virtualization technology is the control of network connectivity of virtual machines. Some projects [Smith and Nair 2005,Keahey et al. 2004, Emeneker et al. 2006]apply virtualization to create a virtual local area network containing resources that are physically far from each other, belonging to different administrative domains.

It is also possible, by using virtualization, to achieve the opposite approach, namely creation of a large-scale controlled environment with few local physical resources.

In this case, network management techniques might be applied to configure not only the virtual environment itself but also real network characteristics.

A set of virtual machines can be configured to compose subnets communicating with each other through controlled routes and links. In this work, SNMP agents are used to configure network components, and TBF/NetEm [Evans and Filsfils 2007, Hemminger 2007] is used to supply bandwidth control. In order to improve network management of the virtual environment, a module is responsible for the reception of both the real environment description and the desired virtual network configuration, described in *XML* language. The module applies the desired configuration in a running set of virtual machines. At the end of the process, the nodes belonging to the virtual environment are ready to communicate according to the specified rules.

Experiments were conducted in order to evaluate the efficiency of our approach in setting virtual network parameters as well as to check the communication configuration made in the virtual machines that compose the virtual environment. This work complements previous results [Calheiros et al. 2010, Calheiros et al. 2008], where tests were executed in a broader context with the use of network management mechanisms described in this paper.

This paper is organized as follows. Section 2 presents basic virtualization concepts. Section 3 presents the motivation for this work and the context where it is placed. Section 4 presents development of the virtual network manager. Section 5 presents the evaluation of the network manager and Section 6 concludes the paper.

2. System Virtualization

System virtualization is applied not only in many production systems, but also in research from several computer science areas. Nevertheless, it is not a new concept: the first virtual machines were developed in the sixties in order to improve efficient in the usage of computers available by that time [Creasy 1981]. This technology allows simultaneous access to the virtualized hardware by several users, which does not realize that the physical machine is not exclusively available for their use. In this context, virtualization can be understood as a technology that abstracts the hardware, allowing execution of several instances of an operating system in the same physical machine [Smith and Nair 2005].

Virtual machine is a software layer that duplicates a real machine, acting between the hardware and the operating system. It is worth noting that every virtual machine located on a certain real machine is isolated from the others. Thus, the processes executing in a specific virtual machine are independent from the others, and cannot interfere in the processes running on other virtual machines. This characteristic allows for increasing the computer system reliability.

Control and creation of virtual machines are performed by the software known as Virtual Machine Monitor (VMM) or hypervisor. The VMM purpose is to make a specific interface available to each virtual machine, providing a virtual environment similar to a real machine.

Paravirtualization is a technique where the VMM does not completely virtualize the underneath hardware. Thus, the operating systems running on virtual machines have to be adapted, in order to become aware of the presence of the hypervisor. On the other hand,

hardware virtualization is common in many hardware components allowing a controlled direct access to the hardware by VM's operating systems [Neiger et al. 2006, van Doorn 2006]. An example of a paravirtualizator is the Xen Virtual Machine Monitor [Barham et al. 2003], which is used in this work, and is presented in the following section.

2.1. Xen VMM

Xen VMM is an open source project developed by researchers of University of Cambridge. Xen supports x86 platform, making use of the paravirtualization concept.

Following the model of the x86 platform protection levels, Xen has direct access to the hardware and executes in the mode of higher privilege. In the level below, the monitor executes a virtual machine (VM) called `dom0`. This special machine is in charge of starting and closing the other virtual machines, called `domU`. It must be highlighted that the `domU` virtual machines have no privileges accessing or using system components, such as I/O devices, memory, and CPU time.

2.2. Xen Network Architecture

Xen network architecture presents a set of real and virtual components. Xen `dom0` is responsible not only for managing these components but also for supporting network communication among VMs. Virtual switches are used to interconnect both real and virtual interfaces. A virtual machine can have one or more virtual interfaces interconnected by one or more virtual switches. They are managed by the firewall software that intercepts network packages and handles it. These virtual switches can run in one of three modes: bridge, router, or NAT [Xen Source Inc 2007].

Bridged mode is used in this work. In this mode, *iptables* [Welte 2006] rules are used to redirect network packages between real and virtual interfaces. These components can be managed through the `dom0` on each Xen host. This machine natively supports execution of all management softwares.

Management tools such as *iptables* and *xm* (for management of Xen virtual machines) support creation of network environments using virtual machines on a easier way. From this standpoint we can see that VMs can be used as normal hosts in order to, for example, emulate distributed systems. Next section presents a proposal of a virtual environment, based on virtualization techniques, to emulate distributed systems.

3. Automated Emulation Framework

This work is part of a project aimed at implementing a virtualization-based emulation framework (Automated Emulation Framework—AEF) on which a user performs experiments in a virtual distributed system [Calheiros et al. 2010, Calheiros et al. 2008]. The system organization is presented in Figure 1.

In the bottom level of the architecture there is a cluster of computers running a Virtual Machine Monitor. Cluster machines run the emulator software, which uses VMM capacities to build the emulated distributed system that runs user's experiments.

The system input is an XML description of system resources (both virtual and physical) and the distributed experiment (e.g., a grid experiment, a P2P protocol test, an enterprise application, among others). Description of the virtual system includes one or

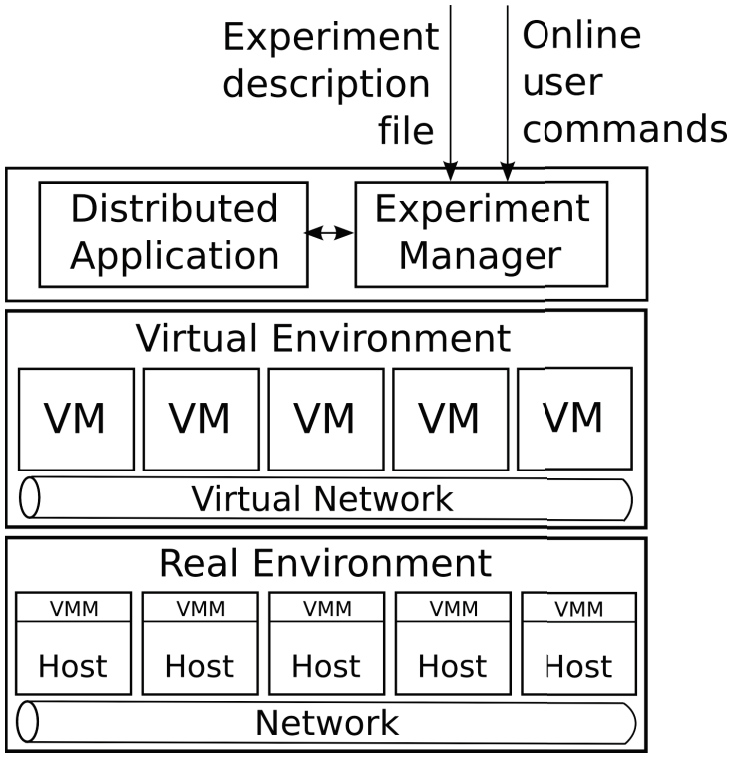

Figure 1. AEF architecture.

more networks, hosts belonging to each network, its configuration, and network characteristics. Description of physical system includes information about the real environment: network, specification of each machine, version of the virtualization software in use, as well as amount of physical resources in use by the VMM on each machine.

Once the required environment is specified, AEF performs a mapping of the virtual resources to real ones, deploys the equivalent virtual system, and configures the network environment in order to supply user demands. Afterward, the user application is automatically triggered in the virtual system and managed by the testbed. Application output is stored according to the user specification.

In this paper, we focus on the aspects related to network management of AEF. In the next section, a description of the network manager implementation is presented, and the experiments and achieved results are presented in Section 5.

4. Virtual Machines Networking

This section presents issues related to network management and configuration used by AEF, as presented in Section 3. Firstly, information about network components of our virtual environment and how they can be managed will be presented. Following, network management aspects applied in this work are presented. Afterward, a software module structure for network management and how this module performs configuration over network components to create a desired virtual distributed environment will be discussed.

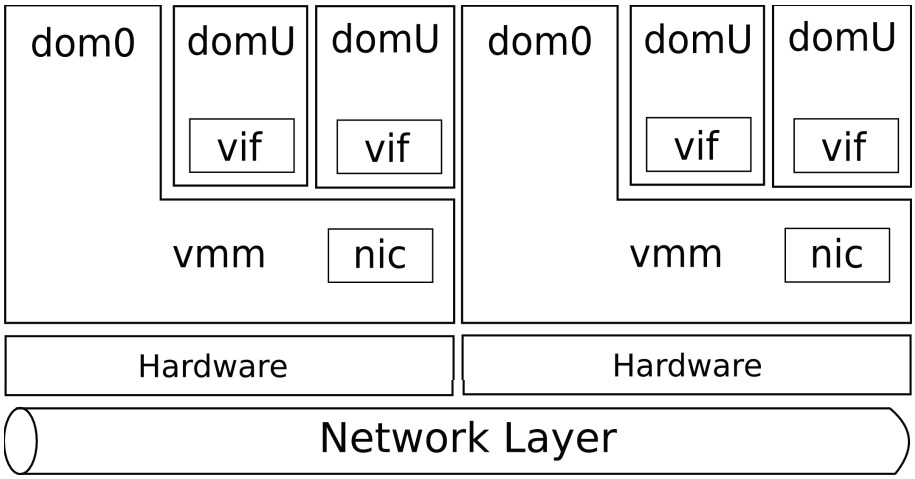

Figure 2. Network of Virtual Machines.

4.1. Network issues

System virtualization technology supports VM network communication by a set of real components and a set of virtual components. As presented in Section 2, each VM has one or more virtual network interfaces (vifs). These vifs are connect in a virtual switch hosted in a privileged virtual machine (in Xen, this VM is called *dom0*). This virtual switch is managed by firewall software that rules all network traffic in the same way that in non-virtualized systems. On this virtual switch, real interfaces are also connected, allowing remote communication. Xen's *dom0* uses *iptables* [Welte 2006] to manage network connections among local and remote VMs. In this work all network rules act on OSI's network layer [Day and Zimmerman 1983]. Bandwidth control is made by a Token Bucket Filter (TBF) [Evans and Filsfils 2007] and latency configuration is performed with the NetEm [Hemminger 2007]. Another bandwidth control algorithms like CBQ [Floyd and Jacobson 1995] and HTB [aka devik] were evaluated, but TBF was used because it provides all the tools required by this work.

At VM start up, IP address and default network routes are set to each vif. Such a definition enables network communication among VMs. However, other network parameters such as bandwidth and latency are set in the firewall software running in *dom0* after VMs start up. The virtual switch transmits all network packages on demand to VMs located at the same host, using token ring algorithm and memory reallocation [Xen Source Inc 2007]. Package transmission between VMs placed in different hosts is done by a real network interface in the same way that in non-virtualized network communications. Figure 2 illustrates network components architecture in a virtualized environment with more than one host.

Almost all network configuration parameters, such as vif's IP address and routes, traffics rules, and bandwidth control can be changed on the fly. A good way to configure a large-scale environment based on these technologies is through a network manager approach. In this work, Simple Network Manager Protocol (SNMP) is used for manage both real and virtual network components. It is presented in the next section.

4.2. Network Management

Network management is used in different systems to configure different kind of resources. SNMP (Simple Network Management Protocol) was originally developed for network management. However, due to its flexibility, it is also used for others management types, such as software management. The structure of this protocol is based on managers and agents, where the agents are spread on the resources and the management operations are performed through direct solicitation from managers to agents. The objects that can be managed are described in the MIB (Management Information Base) [Stallings 1999].

In this work, we apply this approach to configure a environment based on virtual machines. Once manageable components are defined, the environment configuration is performed by a SNMP manager. Network configuration in this work consists basically in making changes on virtual machines network configuration, defining traffic rules on `dom0` (responsible for VMs communication), and setting both bandwidth and latency to any network link among virtual machines. Both bandwidth and latency configurations are essential to create a virtual communication environment as close as possible of network connections in real worlds. Common agents are used to configure network components like network traffic rules, bandwidth, and latency. On the other hand, specific agents to make changes on virtual machine network components were developed to support basics configurations. These agents were described in a previous work [da Cunha Rodrigues 2008].

As presented in Section 3, the proposed Virtual Environment is created in an automatic way. Thus, the network management needs to be automatized to allow easy environment creation. To improve the network management, a module was defined using features presented in this section. This module configures the network communication among VMs and it is described below.

4.3. Network Manager Module

The Network Manager Module was contextualized in Section 3 and it will be described in this section. It was developed as an independent module, and can be used out of the presented context, e.g., configuring network issues of a set of virtual machines described on the input *XML* files. The Network Manager Module is an object-oriented program developed to manage network components through the SNMP protocol. It is used to perform configuration in an environment based in virtualization technology.

Figure 3 illustrates the Network Manager Module. It receives two XML files as input. One of them describes the real network which contains description of routers, physical network links, on-line virtual machines, and network information of real machines. The second XML file describes the desired virtual environment and contains information regarding subnetworks of virtual machines, bandwidth and latency among VMs, network routes and rules, and so on. It is worth noting that this information is a subset of the information supplied by the user of AEF.

The files are mapped in an object-oriented structure by the Network Manager Module. Real network description file is the first file loaded by the Network Manager Module. After all information is read, the module loads the virtual environment file description. All information is used to perform objects creation. To each kind of component

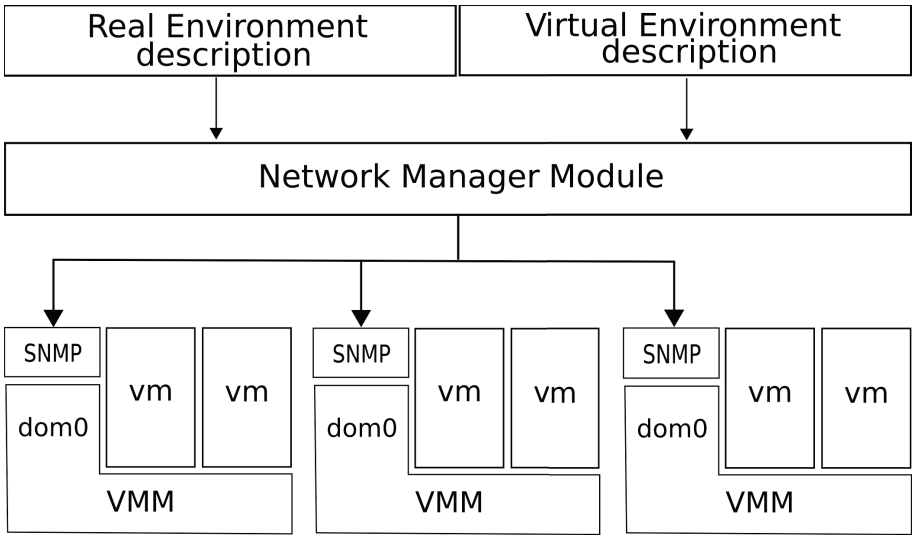

Figure 3. The Network manager module.

loaded from the file, a related object is created with the information needed to perform future configurations in the environment.

The module has a method called *apply* that, when invoked, reads the objects configuration and apply it to the real component which corresponds to the given object. These configurations are made through SNMP protocol as pointed before. The Network Manager Module can run in any machine at the same network of Xen host machines. Each machine needs to run SNMP agents and support remote access to allow module management.

SNMP agents are installed on each Xen host, allowing both configuration and monitoring of VMs using Xen API. These agents are invoked by the Network Manager Module to configure the environment based on the merge between the real and the desired environment description. SNMP agents are responsible for configuring networking of VMs and making changes in `dom0` firewall software where configuration in bandwidth, latency, and traffic rules is made, as shown in Section 2.

After configuration file loading, the module makes changes on vifs of each VM and configures the virtual switches where configuration like network isolation, bandwidth limitation, and other traffics rules are defined.

At the same time, agents offer monitoring capabilities through SNMP *get* command. Agents, located on all Xen hosts, collect information that can be read by a SNMP manager. Network status, bandwidth, and latency are some of information that can be monitored.

Network management using SNMP was considered a good way to manage virtual network components because SNMP is a consolidated and well-accepted management protocol.

5. Evaluation

In order to investigate the capabilities of our approach for VM networking, an experimental testbed using Xen VMM [Barham et al. 2003] was built. In such an experiment,

a grid infrastructure encompassing 3 isolated sites was set and emulated in a cluster of workstations, as described below. The tests presented in this section are qualitative and used to demonstrate network management capabilities of our proposed Network Manager Module.

5.1. Physical environment

The physical infrastructure hosting the virtual environment is a cluster composed of 4 machines. Each machine is a Pentium 4 2.8GHz with 1MB of cache and 2.5GB of RAM memory. The machines in the cluster are connected by a dedicated switch and a Fast Ethernet network. Machines have Xen VMM 3.1, and the Xen's Dom0 uses 328 MB of the available RAM memory. The VM images are stored in the hard disks of each host. Each physical machine hosts more 8 VMs, each one using 256MB of RAM memory and sharing the same amount of CPU time. Only the network traffic generated by this experiment was present in the physical environment during the tests.

5.2. Virtual environment

The virtual environment built to our tests is composed of three subnetworks (sites), each one containing a proxy which is connected to the other proxies. Each subnetwork has a different number of hosts. Physically, the VMs that belong to these subnetworks are distributed among the 4 real machines, as shown in Table 1. In such a table, each line represents a physical machine, and each column represents a site. Thus, a given name in a table cell means that the hosts belongs to the site related to its column and is physically located in the host represented by its line.

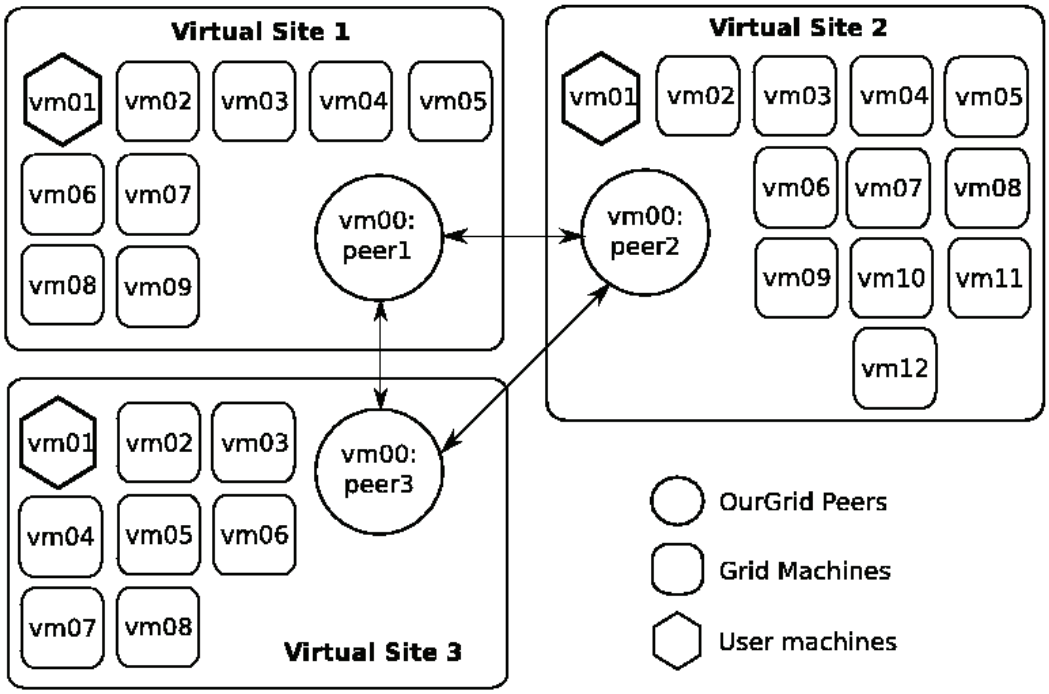

Figure 4. Virtual grid environment built for the tests.

Table 1. Distribution of virtual machines among physical hosts and virtual sites.

	virtual site 1	virtual site 2	virtual site 3	total
host 1	vmgrid108, vmgrid109	vmgrid201-vmgrid205, vmgridpeer2		8
host 2		vmgrid206-vmgrid212	vmgrid308	8
host 3	vmgrid101-vmgrid107, vmgridpeer1			8
host 4			vmgrid301-vmgrid307, vmgridpeer3	8
total	10	13	9	32

Table 2. Time in milliseconds for task file transmit.

Bandwidth scenario	Sending	Receiving
1Mbit/s	136346	113167
100Mbit/s	5197	3463

As shown in Table 1, each machine hosts 8 VMs, belonging to one or more virtual sites. Even though, no direct access was possible among VMs hosted in the same machine but belonging to different sites, because Xen offer isolated virtual machines [Barham et al. 2003] and no routes are set among different subnets. This configuration shows that it is possible to build isolated virtual domains using Xen.

The machines labeled `vmgridpeerX` on each virtual site acts as proxies, having access to the other proxies through the network routes table. Each VM belonging to a site has access to the other VMs belonging to the same site, even though being located at different physical hosts. Figure 4 shows the virtual Grid environment and presents site isolation and proxies communication.

5.3. Network validation

The Grid jobs that ran in our evaluation copies one 1MB file to a grid node twenty times. To evaluate and show network traffic control effectiveness, two bandwidth scenarios were considered: limited in 1Mbit/s among grid proxies and unlimited (using all network capability, 100Mbit/s). Table 2 shows times needed by the grid scheduler to send and receive the file from the grid resource.

Each job has twenty tasks and each task copies the file to the node, runs instructions, and copies the file back. The job was initialized in the user machines, shown in Figure 4, and configured to execute on nodes in remote sites. In this environment only communication among grid schedulers was limited. Although an analysis of the transfers shows a disparity between the two bandwidth scenarios, it is important to consider effects of Ethernet collisions caused by communication between the grid scheduler and the grid resources. Furthermore, in a virtualized environment there is an overhead to transmit

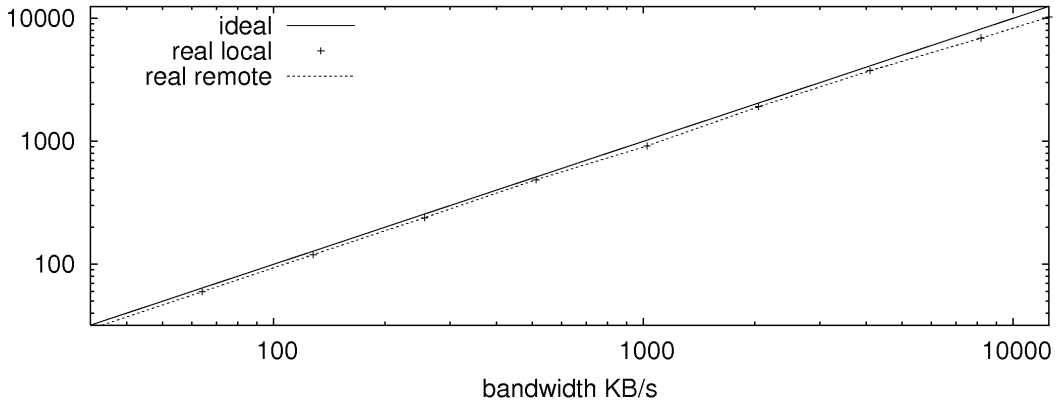

Figure 5. Limited network bandwidth between local and remote VMs.

network packages among distributed virtual machines.

In order to evaluate the effectiveness of our network bandwidth control, a sequence of tests were made. In these tests, we considered both communication between VMs hosted at the same host and communication between remote VMs (i.e., VMs hosted in different machines).

Bandwidth control was made by the TBF (Token Bucket Filter) using *tc* (Traffic Control Linux tool). The configuration was performed in the virtual interface (on Xen `dom0`) related to each VM network interface.

In our tests 300MB file was transmitted in each case (local and remote VMs), and the bandwidth between sender and receiver varied on each test. Secure Copy Protocol [Barrett and Silverman 2001] was used to transmit the file among the nodes.

Figure 5 shows the result of these tests. The straight line represents configured values for the bandwidth, while the dotted lines represent the values obtained in the measurements. The proportionality between the configured value and the measured value increases as the desired bandwidth increases, both in local and remote communication. In a distributed environment, low values of bandwidth are more common than high values, so in the most experiments this approach for bandwidth control will produce satisfactory results.

In order to evaluate the effectiveness of latency control we also varied the latency between sender and receiver in a different test.

Latency control was made by the NetEm (Network Emulation) using *tc* (Traffic Control Linux tool). The configuration was performed in the virtual interface (on Xen `dom0`) related to each VM's network interface.

ICMP echo/request packages were transferred between both local and remote VMs. Figure 6 shows the configured latency (straight line) and the obtained latency (dotted lines). It is possible to see that obtained low latency values have a higher deviation from the configured value, and the proportionality decreases as the latency value increases. In distributed environments, high values of latency are more common than low values, so in most cases this approach also produces satisfactory results.

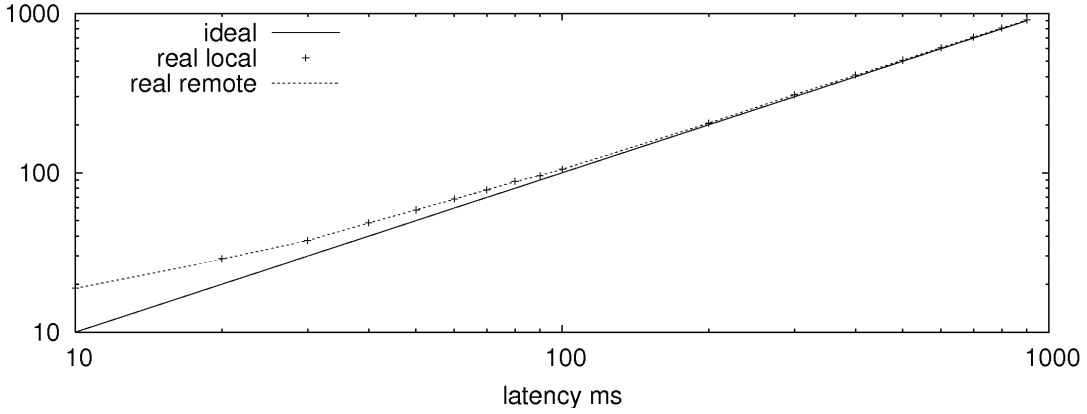

Figure 6. Limited network latency between local and remote VMs.

In virtualized emulation environment, control of bandwidth and latency is an important element to enable simulation of distributed systems. Thus, results obtained in the experiments show that the use of common network tools combined with paravirtualization technology allows a reliable emulation of a distributed system, including network links among nodes.

6. Conclusion and Future Work

Among many possible applications for system virtualization which emerged in the last few years, emulation of distributed environments is a topic that can be further investigated by researchers. When using virtualization in this context, network management is the most important issue to be considered.

In this work, we investigated the use of virtualization to support reliable network configuration of a virtual controlled environment based on virtualization technology. To automate such a process, characteristics of the components were studied and a network management module was implemented. This module receives descriptions of the real environment and the desired virtual network. By using available network tools, such as TBF/NetEm and *iptables*, combined with VM management tools such as *xm*, it was possible to create isolated subnetworks with controlled bandwidth and latency between their links.

To validate our approach, quantitative tests were performed and resulted in similar results between configured and obtained values in two cases: communication among nodes in the same host and communication among nodes in remote hosts. Therefore, we believe that the use of virtualization technology is an interesting alternative for distributed systems emulation, allowing easy configuration of a controlled environment at low cost using few machines to emulate a large distributed system.

In future work, we expect to increase accuracy and reliability of the emulation environment by applying newer versions not only from the Xen VMM itself but also from other deployed tools. We are also investigating network control in other virtual machine monitors, such as VMware [Devine et al. 1998] and KVM [Harper et al. 2007].

References

aka devik, M. D. Hierachical token bucket. `http://luxik.cdi.cz/~devik/qos/htb/`.

Barham, P. et al. (2003). Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*.

Barrett, D. J. and Silverman, R. (2001). *SSH, The Secure Shell: The Definitive Guide*. O'Reilly & Associates, Inc.

Calheiros, R. N., Buyya, R., and De Rose, C. A. F. (2010). Building an automated and self-configurable emulation testbed for grid applications. *Softw. Pract. Exper.*, 40:405–429.

Calheiros, R. N., Storch, M., Alexandre, E., Rose, C. A. F. D., and Breda, M. (2008). Applying virtualization and system management in a cluster to implement an automated emulation testbed for grid applications. In *Proceedings of the 2008 20th International Symposium on Computer Architecture and High Performance Computing*, pages 97–104, Washington, DC, USA. IEEE Computer Society.

Creasy, R. J. (1981). The origin of the VM/370 time-sharing system. *IBM Journal of Research and Development*, 25(5):483–490.

da Cunha Rodrigues, G. (2008). vMIB : uma MIB genérica para gerenciamento de recursos virtuais. Master's thesis, PUCRS, Fac. de Informática.

Day, J. D. and Zimmerman, H. (1983). The OSI Reference Model. In *Proceedings of the IEEE*, volume 71, pages 1334–1340.

Devine, S., Bugnion, E., and Rosenblum, M. (1998). Virtualization system including a virtual machine monitor for a computer with a segmented architecture. US Patent.

Emeneker, W., Jackson, D., Butikofer, J., and Stanzione, D. (2006). Dynamic virtual clustering with Xen and Moab. In *Frontiers of High Performance Computing and Networking – ISPA 2006 Workshops*, volume 4331 of *Lecture Notes in Computer Science*, pages 440–451, Sorrento. Springer.

Evans, J. W. and Filsfils, C. (2007). *Deploying IP and MPLS QoS for Multiservice Networks: Theory & Practice*. Morgan Kaufmann.

Floyd, S. and Jacobson, V. (1995). Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386.

Harper, R. A., Day, M. D., and Liguori, A. N. (2007). Using KVM to run Xen guests without Xen. In *Proceedings of the Linux Symposium*, pages 179–188, Ottawa. Linux Symposium Inc.

Hemminger, S. (2007). NetEm. http://linux-net.osdl.org/index.php/Netem.

Keahey, K., Doering, K., and Foster, I. (2004). From sandbox to playground: Dynamic virtual environments in the grid. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 34–42, Pittsburgh. IEEE Computer Society.

Neiger, G., Santoni, A., Leung, F., Rodgers, D., and Uhlig, R. (2006). Intel Virtualization Technology: Hardware support for efficient processor virtualization. *j-INTEL-TECH-J*, 10(3):167–177.

Smith, J. E. and Nair, R. (2005). *Virtual Machines: Versatile platforms for systems and processes*. Morgan Kauffmann, San Francisco.

Stallings, W. (1999). *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison Wesley, Reading, 3 edition.

van Doorn, L. (2006). Hardware virtualization trends. In *VEE 2006: proceedings of the Second International Conference on Virtual Execution Environments*, pages 45–45.

Welte, H. (2006). What is netfilter/iptables? http://www.netfilter.org.

Xen Source Inc (2007). Xen interface manual. `http://www.cl.cam.ac.uk/research/srg/netos/xen/readmes/interface/interface.html`.