Towards Self-Managed Adaptive Emulation of Grid Environments*

Rodrigo N. Calheiros[1,2], Everton Alexandre[1], Andriele B. do Carmo[1],
César A. F. De Rose[1], Rajkumar Buyya[2]

[1]Pontifical Catholic University of Rio Grande do Sul
Porto Alegre, Brazil
[2]**Gr**id Computing and **D**istributed **S**ystems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia

{rodrigo.calheiros, everton.alexandre, andriele.carmo, cesar.derose}@pucrs.br,
raj@csse.unimelb.edu.au

Abstract

Distributed systems emulators built with the aid of virtualization tools allow testing of systems in a testbed whose number of real elements are orders of magnitude smaller than the number of virtual elements being tested. However, to allow testers to benefit from these systems, operation of the virtual environment should be hidden from them and performed automatically by the emulator. Moreover, testers may be unsure on the exact needs of their environment, and thus can request an environment that does not fit the experiment. In this paper we present our achievements in providing an emulation framework able to provide environment reconfiguration if the requested one does not comply with experiment's demands. Also, it supplies services such as execution log, environment monitoring, and automatic management of applications running in the virtual environment.

1. Introduction

Although research and development in subjects such as grid computing [8], utility computing [13], and cloud computing [3] has grown noticeably in the last years, testing of applications for these distributed environments is still a challenging task. This is mainly because these topics demand access to distributed resources where no entity has control over all the components of the system. Moreover, availability of these resources can vary along the time. This lack of control and access over third-party resources limits both the cases that can be covered during the test and the possibility of reproduction of tests.

To overcome such limitations of tests in real environments, several distributed system emulators built upon virtualization technology were proposed [1,4,5,7,9,12]. These projects aim at delivering a scalable and controlled testbed where a tester can evaluate the behavior of the actual system. The main advantage in using virtualization to develop emulation tools is that it allows a simpler implementation of the emulator because issues related to host resources (e.g., memory, CPU time) multiplexing and network multiplexing are performed by the virtual machine monitor (VMM). A VMM enforces multiplexing of resources by managing the distribution and sharing of resources among isolated virtual machines (VM).

A major drawback in the effective use of emulators to test distributed systems relates to usability, since most approaches require that testers directly operate the platform. As a result they must know how to use virtualization tools to configure both virtual machines and the virtual network. Also, testers must specify the amount of resources required by each entity. However, testers may be unsure on the exact needs of their environment, which can lead to a poor choice of VM configuration parameters. Bad choices compromise the scalability of the experiment (due to overestimation of VM and/or network requirements) or cause concurrency effects because of the lack of physical resources (due to underestimation of VM and/or network requirements).

To circumvent these issues, we present in this paper an architecture that enables self-managed emulation of distributed systems. Our architecture also enables the dynamic reconfiguration of the emulated environment during the execution of the experiment in order to get the most effective

*This research was partially supported by CAPES PDEE research grant 1185-08-0.

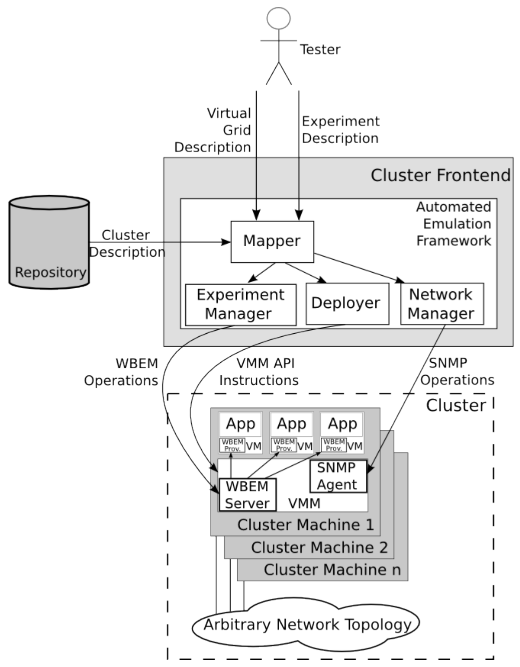


Figure 1. Automated Emulation Framework architecture.

use out of the available physical resources. It implements services such as an execution log, environment monitoring, and automatic management of applications running in the virtual environment. These features run automatically in a cluster of workstations.

2. Automated Emulation Framework

The work presented in this paper is part of the Automated Emulation Framework [4], whose architecture is presented in Figure 1. The target architecture of the framework is a cluster of workstations. The control software runs in the cluster frontend. Cluster nodes can be either homogeneous or heterogeneous, and they can be connected by any network topology. The Virtual Machine Monitor software is required only in the cluster nodes, and all the nodes must run the same version of it. Each module of the framework interacts with one component of the virtualized system.

The system receives from tester as input descriptions of the virtual environment and of the experiment. Cluster description is known by the system and includes information about network (e.g., cluster network topology), specification of each machine (e.g., their capacity, amount of memory available, and network addresses), version of virtualization software in use, and amount of physical resources in use by virtual machine monitors on each machine. Description of the virtual environment supplied by the tester

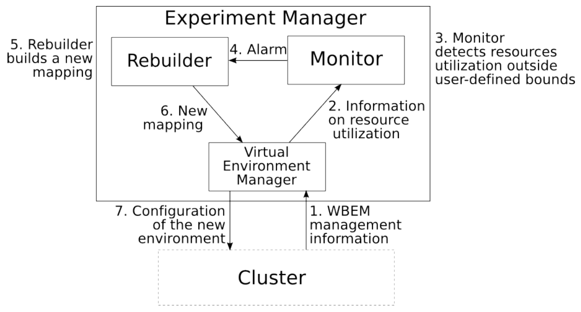

Experiment Manager
5. Rebuilder builds a new mapping
Rebuilder
4. Alarm
Monitor
3. Monitor detects resources utilization outside user-defined bounds
6. New mapping
2. Information on resource utilization
Virtual Environment Manager
7. Configuration of the new environment
1. WBEM management information
Cluster

Figure 2. Self-managed adaptive infrastructure. Boxes represent modules. Also, it is shown the role of each component in the reconfiguration process.

contains the nodes in such environment, their configuration, and configuration of connections between elements. Description of the experiment contains applications that run on each node, its parameters, and lower and upper bounds of resource usage by both virtual and physical resources.

In the environment building cycle, the Mapper module uses the tester input to determine both where each VM will run and the physical path that will correspond to each virtual link between virtual nodes. This mapping is used by the Deployer and Network Manager to create the virtual environment and configure the network, respectively. Finally, the Experiment Manager, using the Web-Based Enterprise Management (WBEM) resource management specification [10], manages the execution of the experiment, according to the description supplied by the tester.

3. Self-managed Experiment Execution

Originally, the Automated Emulation Framework could only build the environment in a static way. Also, its Experiment Manager could neither manage applications running in the virtual machines nor monitor network and resources conditions. The architecture presented in this paper, which is an enhancement in the Experiment Manager Module, allows the system to support not only automatic reconfiguration of the environment if it does not behave the way tester determined in the experiment description but also automatic execution of the experiment, by triggering applications inside each VM.

To deliver the features presented previously, a self-managed adaptive infrastructure is proposed. This infrastructure delivers three core services, namely environment management, alarms triggering, and environment rebuilding. These services are provided by three modules. These modules and its relation to the reconfiguration cycle are presented in Figure 2 and are detailed next.

3.1. Virtual Environment Management

The Virtual Environment Management is the service from the self-managed adaptive infrastructure that acts directly on both the physical and virtual environments. The Virtual Environment Manager Module provides this service. This module supplies services to monitor and control the life cycle of virtual machines. Additionally, it provides services to control applications that are executed inside the virtual machines.

Figure 3 presents a detailed architecture of the Virtual Environment Manager Module. It has two main components, the Service Management running in the frontend, and the Environment Management running atop the Virtual Machine Monitor.

The first main component, the Service Management, offers a group of services to other modules from the infrastructure. This component receives requests from the higher-level modules and invokes the Environment Management located on each cluster node. Services of this component are accessed by other components through a Java API. This API offers methods such as `getVMMemory()` and `getVMCPU()` to answer queries related to usage of virtual machine resources.

Internally, these queries correspond to services processed with the use of the WBEM specification [10]. Each service of this component has an associated WBEM client intended to perform requisitions to the WBEM servers present in each Manager of Virtual Environments. Such requisitions can represent either a request of any information, such as the amount of CPU used by a certain virtual machine, or the invocation of a command, such as the creation of a virtual machine.

The second main component, the Environment Management, runs in the privileged level of the VMM of each host of the cluster. It is composed of a WBEM server that responds to client requests, and a set of local and remote providers that manage specific features. Local providers manage features of their respective host and also perform virtual machine management operations, while Remote providers are installed inside each virtual machine and manage applications running on top of it.

3.2. Alarms Triggering

Monitoring of resource usage in both physical and virtual environment is important to detect violations in the utilization of resources. Once such an exception is detected, an action must be taken in order to reconfigure the environment and avoid further problems in the system. This is done by the Monitor Module, which deals with them through a mechanism of alarms triggering. This module has two purposes. The first purpose is to supply for tester information about usage of resources from hosts, virtual machines, and network. This information is delivered as execution logs after experiment execution, and it is stored in a system repository. The second purpose of this module is to generate alarms to the Rebuilder Module warning it about violations in the resources usage, according to rules defined by the tester. For example, the tester may determine that the bandwidth of a given virtual link should not exceed a specific value. In this case, if the value is exceeded, an alarm is generated and sent to the Rebuilder module, through a Java API, which can decide to start a system reconfiguration.

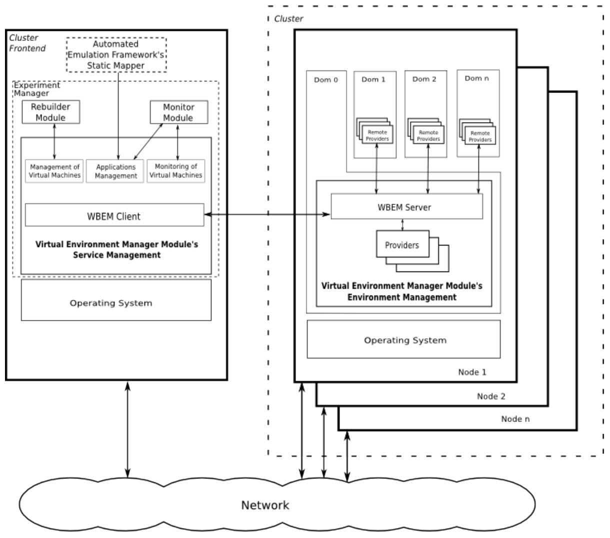


Figure 3. Virtual Environment Manager Module.

Alarms can be caused by events related to physical components of the infrastructure, events related to virtual components of the infrastructure, or both. A unique number that is a power of two is assigned for each different condition that can trigger an alarm (e.g., CPU overutilization, link overload, and so on). The use of powers of two facilitates the identification of combined events by the Rebuilder Module. There is also the possibility that, during the execution of a specific experiment, some of these conditions may be disabled. This is a feature, since only conditions appointed by the tester in the experiment description should generate alarms. The alarm is forwarded to the Rebuilder Module as a message containing both the identifier of the exception and a list of the resources where such violation was observed.

3.3. Environment Rebuilding

The Environment Rebuilding service has two functions. The first one is to determine the characteristics of the virtual environment, e.g., amount of machines and amount of resources allocated to each one. The second function is to map the virtual environment in the real environment, defining where each virtual machine will run and how the virtual links among them will be allocated in the real network.

The module from the Experiment Manager that handles system reconfiguration is the Rebuilder Module. It receives the alarms from the Monitor module, and, considering the specific alarm, defines the action to be taken. The alarms might be set due to resources underutilization (in which case the environment can be scaled up) or due to resources overutilization (which leads to an environment scaling down). Nevertheless, at the same time more than one kind of alarm can be triggered. So, the Rebuilder module must deal with more complex situations than simple "scaling ups" and "scale downs". For example, at the same time, a given set of components can trigger alarms of CPU underutilization and network overutilization. In this case, it is possible that, after increasing network bandwidth, more data will arrive in the application running in hosts, increasing CPU utilization and making the later reach the expected usage. In this example, one action (increase bandwidth) solves two problems in the environment.

The set of actions this module can choose from can be either simple actions, caused by single alarms, or complex actions, caused by the activation of more than one alarm. To deal with these different cases, the Rebuilder sums the value assigned by the Monitor to each alarm. This value, as explained in the previous section, is a power of two. So, a current event is defined by the Rebuilder as the sum of the alarm numbers currently set. For example, the value assigned for memory underutilization is 1 (2^0) and to CPU underutilization is 4 (2^2). Consequently, the event 5 means both memory and CPU underutilization in one or more VMs.

For each event number, there is a list of actions to be taken. If the first element of the list does not solve the problem, the next one is applied. An action consists of a set of instructions for environment changes. It can be one or more adjustments in the amount of resources assigned to virtual machines (e.g., memory). However, these adjustments may require more resources than available in the hosts. Hence, commonly adjustments in resources parameters are followed by changes in the scale of the system. If such modification is required, it is done according to tester's instructions. So, some actions cannot be applied because it might violate the minimum or maximum amount of virtual machines specified by the tester.

To avoid instabilities, e.g., that after a change from a configuration A that overutilizes resources to a configuration B that underutilizes resources the system go back to configuration A, this module contains a mechanism that removes contradictory actions (e.g., actions that performs the reverse of the applied action) from the actions list of the new configuration.

3.4. System Reconfiguration

In this section, it will be detailed how the three previously described components execute the automatic reconfiguration of an emulated environment in order to comply with tester requirements. The information required as input, despite environment and experiment description, is the interval (maximum and minimum values) in which the resources (e.g., memory, bandwidth) must be. With this information, the Monitor Module builds lists to determine which services from the Virtual Environment Manager Module must be accessed and periodically access them to verify system behavior.

Thus, at a regular time interval defined by the tester, each service from the list is invoked, through a specific Java method defined in the module's API. The value received in response is analyzed to verify whether it is within the interval specified by the tester or not. If not, a new alarm is generated, and the element in the system that caused the alarm is associated to it.

After all the relevant information is obtained, elements that caused similar alarms are grouped together to generate a single alarm related to that specific violation (e.g., CPU overutilization). Subsequently, each alarm generated is passed to the Rebuilder Module through the invocation of a Java method.

The Rebuilder receives all the alarms and queries its internal tables to look for the actions related to the specific alarm. Actions are selected according to the policy described in the previous section. A new virtual environment, which contains the modifications proposed by the action, is built and forwarded, via Virtual Environment Manager, to the Mapper from the emulator (shown in Figure 1).

If a new mapping for a given environment is not found, the Rebuilder tries to apply another action from the list. If the actions list is exhausted and either the problem could not be resolved or the proposed new environment could not be mapped, the experiment runs with the last configuration found and a report describing the violations during experiment execution is generated to the tester together with the regular experiment output.

When the new mapping is found, services from the Virtual Environment Manager Module that allow modification of VMs (destruction, migration or change of configuration, depending on the action) are invoked by the Rebuilder Module. These commands are translated in WBEM actions that actually change the environment. When the new environment is ready, the Rebuilder invokes services related to applications triggering in the VMs and the experiment starts again.

4. Evaluation

In this section we present the results of an experiment to show that the Application Manager Module can effectively detect violations in resource usage by applications and react accordingly, creating a new environment able to comply with tester specification.

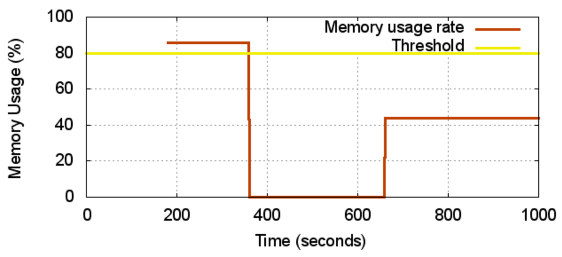
The physical environment consists of four Pentium 4 2.8GHz with 1MB of cache and 2560MB of RAM memory. Cluster machines are connected by a dedicated Fast Ethernet switch. Machines run Xen VMM [2] version 3.1, and Xen's dom0 uses 328MB of the available RAM memory. Thus, 2232MB are available to the guests on each host. No network traffic but the one generated by this experiment was present in the physical environment during the tests.

The experiment input for the framework was a partial description of a local area network. The original configuration proposed by the tester is composed of 32 virtual machines with 256MB of memory each. According to the input rules, the system can be scaled down to 2 machines, and can be scaled up without limit, as long as the use of memory on each virtual machine is kept below 80%.

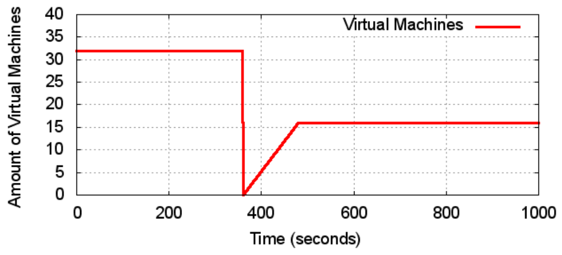
To force the alarm mechanism to be activated, experiment description includes instructions to run an application that allocates 200MB of RAM memory on each virtual machine. With this application, we could assure that memory utilization on virtual machines was above the threshold and thus the alarm and reconfiguration mechanism would be activated. Additionally, we could also observe the mechanism for automatic triggering of applications inside virtual machines in action.

The experiment description and the virtual environment description were supplied to the Automated Emulation Framework. The later built the virtual environment. Then, the Experiment Manager has been invoked to run the experiment. The Experiment Manager, through the Virtual Environments Manager Module, triggered the application in the virtual machines.

Figure 4(a) shows the memory utilization of one virtual machine during the experiment and Figure 4(b) shows the number of virtual machines running in the experiment. Initially, there is no information about resource usage because this data has not been collected in any of the running virtual machines. After 3 minutes (as defined by the tester in this experiment), information about memory utilization is collected. The Monitor Module detects that memory utilization in the virtual machines is above the configured threshold of 80% and triggers an alarm that activates the Rebuilder Module. The first reconfiguration action considered (doubling the amount of memory on each VM) fails because there is no enough physical hosts to accommodate 32 VMs with 512MB of RAM. The second action, doubling the memory and reducing the number of VMs from 32 to 16, allows the



(a)



(b)

Figure 4. (a) Memory utilization of one virtual machine (b) Number of virtual machines in the experiment.

Mapper to find a valid mapping. In the new mapping, 4 virtual machines run in each host. Because a valid mapping was found, reconfiguration procedures were triggered in the system.

After the reconfiguration of the environment, that finished at t=480s, the experiment and the Monitor Module were started again. With this new configuration, the memory usage in the virtual machines (detected 3 minutes later) was below the defined threshold and the experiment finished without further reconfigurations, which is the expected behavior of the system.

5. Related works

Several emulation tools build upon virtualization technologies have been proposed recently. vBET [11] can configure emulated networks automatically in a single host. However, it cannot run in a distributed environment like a cluster, what limits scalability of the environment. V-DS [12] can run static experiments from scripts. However, it does not support tools for running applications inside virtual machines. Other approaches, such as NEPTUNE [5] and V-eM [1] does not support automatic configuration of the environment, so this task is delegated to testers. Test-Grid [7] relies on GridBuilder [6] to deploy the system. It also cannot provide tools to run applications inside a virtual machine. None of these systems provide experiment management and dynamic reconfiguration, as does our approach. DieCast [9] uses a different approach for the emulation, applying a concept of time dilation to overcome restrictions in resources availability.

The same issues considered in our work in the context of grid emulation are addressed by HARMONY [14] in the context of load balancing in storage and computing datacenters. HARMONY provides an environment for monitoring usage of conditions and remapping virtual machines and virtual disks in order to meet SLA criteria from datacenter users. This difference in the usage of the system leads to differences in both approaches: for example, HARMONY must make decision in real time, in order to not violate SLA agreements. Also, all the remapping actions must be performed without interruption in the services. So, decisions of mapping in HARMONY consist in virtual machine migrations only, while our approach consider also changing the amount of resources allocated to the virtual machines.

6. Conclusion and future works

The current availability and reliability of system virtualization software enabled a large range of new applications to this old technology. One of these applications is allowing the building of emulated distributed environments in regular clusters of workstations. The emulated environment can be used to test and evaluate software in a scale hard to be achieved in real environments before deploying it in a real environment.

In this work, we presented an architecture that allows execution logs, automatic management of applications running in an emulated environment, environment reconfiguration, environment monitoring, and dynamic adaptation of the environment. All these services run automatically, which is a differential among the current approaches for the same problem.

Our evaluation demonstrates the effectiveness of the three modules composing the Experiment Manager of our Automated Emulation Framework in not only managing the virtual environment but also controlling applications running inside the virtual machines. Furthermore, this management allowed the detection of violations in the experiment configuration, being a basic requirement for environment reconfiguration.

In the future, we intend to develop more efficient heuristics to map a virtual environment to a network of workstations and investigate the applicability of our adaptive emulation to other distributed execution platforms. We believe that the tools and methodology presented here can be applied directly, or with minor adaptation efforts, in other contexts that require automatic management and adaptation of virtualized environments. The most promising of these contexts is the emerging Cloud Computing [3] platform, in which our solution might be applied to provide WBEM-based management, deployment, and automatic reconfiguration of VMs to the infrastructure.

References

[1] 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.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *19th ACM Symposium on Operating Systems Principles*, 2003.

[3] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities. In *10th IEEE International Conference on High Performance Computing and Communications*, 2008.

[4] R. N. Calheiros, M. Storch, E. Alexandre, C. A. F. D. Rose, and M. Breda. Applying virtualization and system management in a cluster to implement an automated emulation testbed for grid applications. In *20th International Symposium on Computer Architecture and High Performance Computing*, 2008.

[5] R. Canonico, P. D. Gennaro, V. Manetti, and G. Ventre. Virtualization techniques in network emulation systems. In *Workshop on Virtualization/Xen in High-Performance Cluster and Grid Computing*, 2007.

[6] S. Childs, B. Coghlan, and J. McCandless. GridBuilder: A tool for creating virtual grid testbeds. In *2nd IEEE International Conference on e-Science and Grid Computing*, 2006.

[7] S. Childs, B. Coghlan, J. Walsh, D. O'Callaghan, G. Quigley, and E. Kenny. A virtual TestGrid, or how to replicate a national grid. In *15th IEEE International Symposium on High Performance Distributed Computing*, 2006.

[8] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, 1999.

[9] D. Gupta, K. V. Vishwanath, and A. Vahdat. DieCast: Testing distributed systems with an accurate scale model. In *5th USENIX Symposium on Networked Systems Design and Implementation*, 2008.

[10] S. Harnedy. *Web-Based Information Management: An Introduction to the Technology and its Application*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[11] X. Jiang and D. Xu. vBET: a VM-based emulation testbed. In *ACM SIGCOMM workshop on models, methods and tools for reproducible network research*, 2003.

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

[13] M. A. Rappa. The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32–42, 2004.

[14] A. Singh, M. Korupolu, and D. Mohapatra. Server-storage virtualization: integration and load balancing in data centers. In *ACM/IEEE conference on Supercomputing*, 2008.