SOFTWARE – PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2010; **40**:405–429
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe.964

Building an automated and self-configurable emulation testbed for grid applications

Rodrigo N. Calheiros[1,2], Rajkumar Buyya[2] and César A. F. De Rose[1,*,†]

[1]*Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil*
[2]*Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia*

SUMMARY

Distributed systems, such as grids, are composed of geographically distributed computing elements that belong to multiple administrative domains and are controlled by multiple entities. It is unlikely that testers are able to acquire repeatedly the same resources, for the same amount of time, and under the same network conditions, which are paramount requirements for enabling reproducible and controlled tests in software under development. An alternative to experiments in real testbeds is the use of emulation tools, which allow the software to run in an environment that behaves like a distributed system. Although advances in virtualization technology allowed the development of efficient emulators, few efforts were put in making operation of such emulators easier. This paper presents the design and the development of the Automated Emulation Framework that allows automatic mapping of virtual machines to hosts, virtual machine deployment, network configuration, and proactive management and reconfiguration of the virtual infrastructure. Copyright © 2010 John Wiley & Sons, Ltd.

Received 9 July 2009; Revised 5 October 2009; Accepted 4 December 2009

KEY WORDS: emulation; virtual machines mapping; virtualization; systems management; grid computing

1. INTRODUCTION

An important stage of the development of any software is observing its behavior when executing in the software's target environment, or in an environment that resembles the target environment. In the case of distributed systems, however, this task is harder than in the conventional systems.

The reason is that it is difficult for software testers to obtain access to computing nodes in an amount that is satisfactory for comparing with a real production environment. Distributed systems, such as grids [1], are composed of computing elements that belong to multiple administrative domains, are geographically spread, and are not controlled by a single entity. Hence, it is unlikely that testers are able to acquire repeatedly the same resources, for the same amount of time, and under the same network conditions, which are paramount requirements for enabling reproducible and controlled tests.

Access to resources is not required in the earlier stages of software development. Simulation tools [2] can be used in such stages to validate algorithms and protocols prior to software development.

*Correspondence to: César A. F. De Rose, Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Av. Ipiranga, 6681—Partenon—Porto Alegre/RS Brasil CEP 90619-900, Brazil.
†E-mail: cesar.derose@pucrs.br

Contract/grant sponsor: CAPES PDEE; contract/grant number: 1185-08-0
Contract/grant sponsor: Australian Research Council (ARC)
Contract/grant sponsor: Department of Innovation, Industry, Science and Research (DIISR)

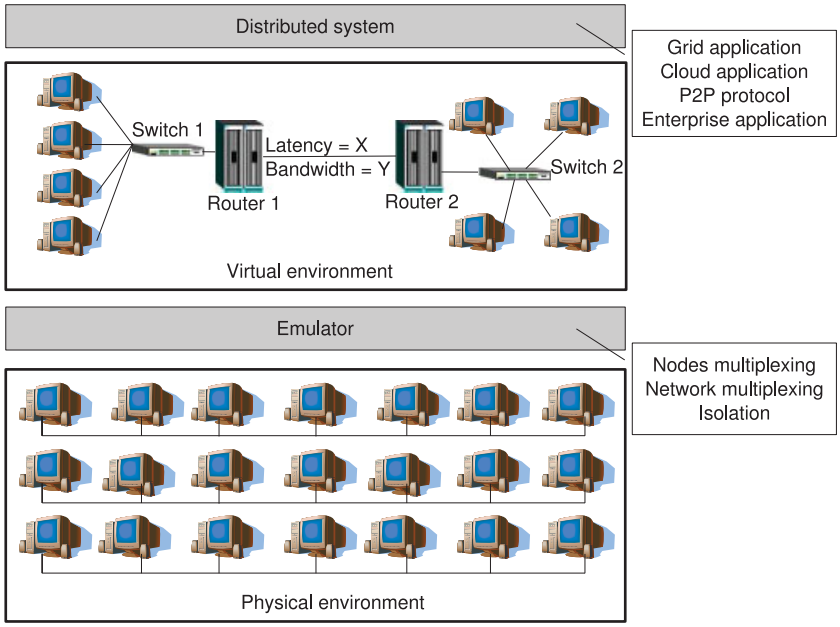

Figure 1. Overview of a general distributed system emulator.

However, when a software prototype is already available, simulation is not a suitable approach, because these tools require abstraction of both specific software characteristics and low-level details of the environment in order to represent high-level aspects of the system. Then, specific software behavior may be missed when modeling it to the simulator language, which is an undesirable effect.

In the later stages of software development, when a prototype is already available, it is important that testers are able to analyze the effects of network and scale in the software. Also, replication of conditions previously observed is important to identify the problems in the software. Because replication and control in real testbeds are not always possible, alternative options, such as emulation [3], can be used to replace real distributed systems.

A general view of an emulator is presented in Figure 1. At the top level, a distributed software, which may be a grid [1] or cloud [4] application, a P2P protocol or a business system, among others, runs in a virtual distributed environment. The actual infrastructure is a local network, a cluster, or even a single machine that, with the aid of the emulator software, behaves like the desired distributed system.

The drawback of emulators is that they are usually very complex software, because they encompass activities such as processor multiplexing (to emulate multiple computing nodes in a single physical node) and network multiplexing (to emulate multiple and isolated network connections in a single real network connection). Also, they usually require special hardware to adequately host the environment [5].

Development of such software became easier with the development of modern virtual machine monitors (VMMs), such as Xen [6] and VMware [7], which allow the deployment of several isolated virtual machines (VMs) in a single physical host. Virtualization software enables presence of various nodes, running different operating systems with different amount of resources in a single hardware [8].

Although utilization of virtualization makes easier the development of emulators, it does not make easier the operation of emulated environments. That happens because the currently available emulators require that testers configure the virtual environment, either by manually running VMM commands or by using some tool for direct management of virtualized environments. This approach is undesirable for three reasons. The first one is that it demands knowledge on how to operate virtualization tools, which consumes the time that could be used in activities related to the tester

DOI: 10.1002/spe

activities [9]. The second reason is that testers may be unsure about the exact requirements of the emulated environment, which may either compromise the system scalability (because each virtual node has more resources than required by the experiment) or cause system malfunctioning (because of lack of resources in the virtual nodes) [10]. The third reason is that manual operation of the virtual system is impractical for high-scale systems, because the tester may be unable to manually find a valid mapping of eventually hundreds or thousands of VMs to tens or hundreds of hosts [11].

To circumvent all these problems in distributed systems' emulation, the Automated Emulation Framework (AEF) [9, 10] was developed. AEF allows automated emulation of grids and automated execution of experiments in a cluster of workstations.

In this paper, we gather, update, and expand the previous research on AEF. The contributions of this paper are the following:

- it presents the mechanisms to allow automatic mapping of VMs to hosts, VMs deployment, and network configuration;
- it presents the mechanisms for management of both the physical and the emulated infrastructure. This mechanism detects system misbehavior, overload, or underload of the system and proactively reacts, stopping the execution and reconfiguring the system to return to a state defined as acceptable by tester; and
- it presents the formal definition of the mapping problem that allows the automatic mapping of VMs to hosts; also, four mapping algorithms with different goals are presented and compared.

The remainder of the paper is organized as follows. Section 2 presents other works related to the one presented in this paper. Section 3 presents an overview of the AEF, its target environment, and design goals. Sections 4 and 5 detail the modules that constitute AEF. Section 6 presents the evaluation of each part of AEF and Section 7 presents the conclusions and future direction of this research.

2. RELATED WORKS

In this section works that relate to the work presented in this paper are discussed. These works are related to emulation, virtual systems management, and mapping of VMs to hosts.

2.1. Emulation testbeds

Several researchers have developed distributed systems emulation testbeds. Emulab [5] is a well-succeeded project in this area. It uses BSD jail [12] in a network-complex environment to multiplex virtual hosts into physical nodes. AEF, on the other hand, executes in regular clusters, and requires less complex software to work, because it exploits the capabilities of VMM software.

Emulation requires multiplexing of computing nodes and network links to allow the representation of systems with more emulated elements than real elements. Nevertheless, enabling multiplexing is complex, because it requires interaction with the operating system kernel to achieve isolation among the emulated components. Because multiplexing is naturally supported by virtualization technology [8], several emulation testbeds and tools proposed recently were built with the use of virtualization technologies. vBET [13] configures emulated networks automatically in a single host. However, it does not run in a distributed environment like a cluster, what limits scalability of the environment. V-DS [14] runs static experiments from scripts. However, it does not support tools for running applications inside VMs. Other approaches, such as NEPTUNE [15] and V-eM [16], do not support automatic configuration of the environment. Thus, testers must set it up manually or with other virtualization management tools. TestGrid [17] relies on GridBuilder [18] to deploy the system. It also does not provide tools to run applications inside a VM. None of

DOI: 10.1002/spe

these systems provide experiment management and dynamic reconfiguration, as does our approach. DieCast [19] uses a different approach for the emulation, applying a concept of time dilation to overcome restrictions in physical resources' availability.

2.2. Virtual systems management

Management of virtualized environments, which in AEF is applied in the context of grid emulation, is addressed by Singh *et al.* [20] in the context of load-balancing in storage and computing data centers. This solution, called HARMONY, provides an environment for monitoring usage of resources and remapping VMs and virtual disks in order to meet SLA criteria from data center users. This difference in the usage of the system leads to differences in both approaches: for example, HARMONY makes decision in real time and without interruption in the services, in order to not violate SLA agreements. Because in AEF the system is not bound to SLAs, the reconfiguration process stops system execution and allows the utilization of more time-consuming strategies for VMs' replacement.

Khanna *et al.* [21] present a solution for data centers management that contains a subset of the features present in HARMONY. In such an approach, management of the virtualized infrastructure is carried out by the IBM Director software. As in HARMONY, the only reconfiguration action considered is VM migration. These migrations are triggered if the load of the host achieves a threshold set by the system administrator. The goal of the reconfiguration is to minimize SLA violations in the infrastructure. This approach, however, does not consider the bandwidth allocation, and thus performs only a part of the management and mapping performed by AEF.

Usher [22] is an extensible and customizable VM management system for data centers. It allows the creation of virtual clusters of VMs in a physical infrastructure. Usher allows the system administrators to express their own policies to set migration, creation, and destruction of VMs. When a condition defined by the user is observed in the monitoring data, the corresponding action set by the administrator is performed. AEF, on the other hand, allows a combination of policies and events to define complex actions for system reconfiguration. Furthermore, in AEF network events observed in the infrastructure also trigger reconfiguration events.

2.3. Mapping of VMs

The mapping problem addressed in this paper has similarities with problems found in other contexts. *Generic Adaptation Problem in Virtual Execution Environments* (GAPVEE) [23] consists of finding a mapping of VMs to hosts and the mapping of virtual links to paths in order to emulate a local area network in a wide area network, whereas in AEF the opposite problem is addressed, that is, the emulation of a wide area network in a local area network. Moreover, in GAPVEE formulation there is a valid initial state where the solution can start from. This happens because in GAPVEE the goal is to find a new mapping that improves a real application throughput. In the formulation presented in this paper and used by AEF, on the other hand, the goal is to find a mapping starting from a state where there are no VMs mapped. Furthermore, each approach has its own optimization goal. Nevertheless, GAPVEE provided good insights on how the mapping problem could be solved. For example, as the mapping problem proposed here has similarities with GAPVEE, and the latter is shown to be NP-Hard [23], heuristic solutions instead of exact solutions were sought for AEF's mapping.

Other problems related to the one presented in this paper are the *Network testbed mapping problem* [24] and the problem defined by Liu *et al.* [25]. In the first approach, mapping of nodes is performed considering that each machine can receive an amount of virtual nodes determined by testers and no consideration is made about the consumption of resources (CPU, memory, and storage) by the virtual nodes. In the second approach, each host receives only one machine, and the resource constraints are considered during the mapping. These works do not consider that resources of the host could be allocated according to the requirements of each virtual node because these solutions were developed before modern VMMs became widespread. Besides, both solutions have restrictions in the topology of the real environment they can map, which limits their applicability.

DOI: 10.1002/spe

Finally, Singh *et al.* [20] applied a heuristic to map VMs and virtual storage systems in data centers. However, such a solution, called *VectorDot*, was designed to work in a very restricted environment, not in general environment as was AEF's approach.

3. AUTOMATED EMULATION FRAMEWORK

The AEF [9, 10] is an emulation testbed for grid applications. It allows testers to describe the distributed system required by the application and automatically deploys a virtual infrastructure corresponding to such a system in a local cluster. The overall AEF architecture is depicted in Figure 2.

The cluster may be either homogeneous or heterogeneous regarding nodes' configuration. It also can have any network configuration. However, it is required that every node run the same

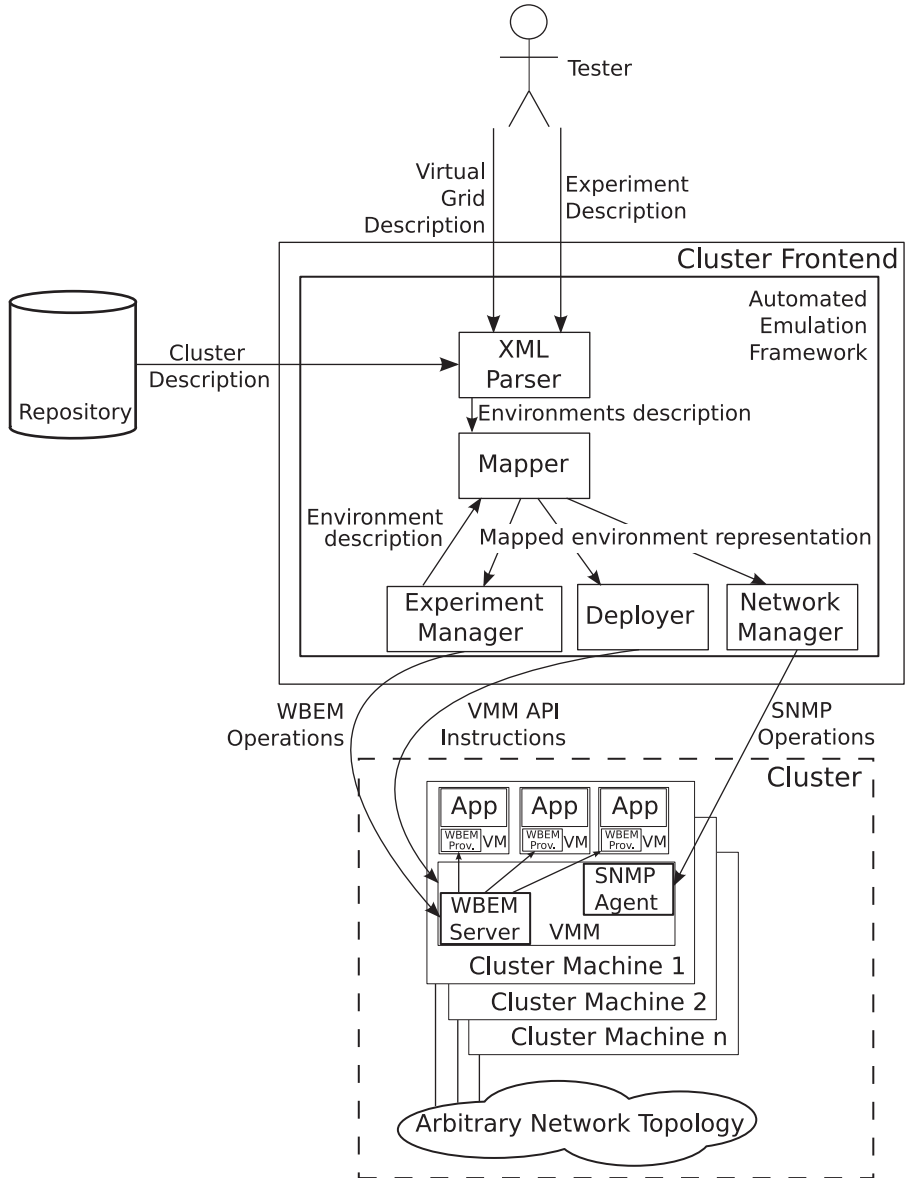

Figure 2. Automated Emulation Framework architecture.

DOI: 10.1002/spe

version of a VMM. The nodes' configuration, the network topology, and application parameters are stored to be used by AEF.

Because the tester may be unsure of the exact requirements of a platform for his or her application, AEF allows a partial description of the environment. In the partial description, the tester specifies, for each emulated grid site, the minimum and the maximum number of machines that are allowed to be deployed. Testers are also able to specify the amount of resources of a site in relation to another site. Hence, it is possible, for example, to say that a given site will have twice the amount of resources of another one. In either case, an initial guess of the number of machines in each site must be supplied by the tester. This initial guess is used in the first deployment of the system, as described later.

Another input from the tester is the limit in the resource usage by VMs and the network links accepted in the experiment. For example, the tester can specify that the use of CPU by VMs must stay below 90%. When this limit is achieved, AEF tries to reconfigure the system in order to reduce the CPU usage, for example by increasing the amount of CPU of the VM. Because some reconfiguration actions may involve changes in the number of VMs in the environment, AEF has to decide which networks will lose or get more VMs. Defining the priorities in changing the number of site resources is also done through the input description file.

The AEF experiment description file is based on the SimGrid [26] description file. It is an XML file containing the number of virtual nodes and their characteristics (name, VM image, amount of memory, CPU, and storage), and details of the connection between the nodes, i.e. whether they belong to the same local network (site) or they represent nodes connected through the Internet. In the latter case, the latency and the bandwidth of such connection is also a system input. A description of the application includes applications to be triggered, node in which they will run, and their parameters. This file is parsed and the information is used throughout the AEF workflow to map and deploy nodes, to configure the network, and to run the experiment.

The Parser module receives the description file and translates it into an internal representation of the network. It also translates the rules related to environment reconfiguration and sites' definition. The network description is forwarded to the Mapper module. The Mapper uses this information and the local cluster description (stored in the AEF) to map both VMs in the cluster and the virtual links to physical paths in the cluster network. Virtual links are mapped to paths because VMs might be placed in hosts that are not directly connected. In this case, the path corresponding to the virtual link passes through intermediate nodes until reaching the destination node. Also, if the VMs are mapped to the same host, the link is handled internally in the host. In this case, the virtual link is not mapped to a physical path.

After the placement of VMs in the cluster is defined, VMs are actually installed by the Deployment module. Then, the Network Manager Module configures the network, creating the virtual routes between VMs in the cluster network according to the mapping supplied by the Mapper module.

When the virtual environment is built, the experiment is triggered and the environment is monitored by the Experiment Manager. To run and monitor the experiment, the Experiment Manager uses the WBEM management protocol [27]. Hence, WBEM servers and agents run together with the VMM in the cluster nodes and in the VMs to enable the system management.

Table I summarizes AEF modules, their goals, and the element in the cluster that they interact with. Details on each of the AEF modules are provided in the following sections.

4. INSTALLATION AND CONFIGURATION

The first stage of AEF operation is the installation and the configuration of the virtual distributed system. It starts after the Parser sends the representation of the physical and the virtual environments to the Mapper module and finishes after VMs' deployment and network configuration. Figure 3 depicts this initial installation and configuration workflow.

Because the amount of VMs to be created in the system may be large, the mapping must be performed automatically. Also, because AEF allows only one tester at a time, the goal of the

DOI: 10.1002/spe

Table I. Summary of AEF modules and their function.

Name	Task	Interacts with
Parser	Parsing of XML input files	Mapper
Mapper	Mapping of virtual environment to physical one	Deployer, Network Manager, and Experiment Manager
Deployer	Installation of VMs in the cluster	VMM (or VMM manager)
Network Manager	Configuration of virtual network	SNMP agent in the VMM
Experiment Manager	Installation, configuration, and monitoring of the virtual environment	WBEM server in the VMM

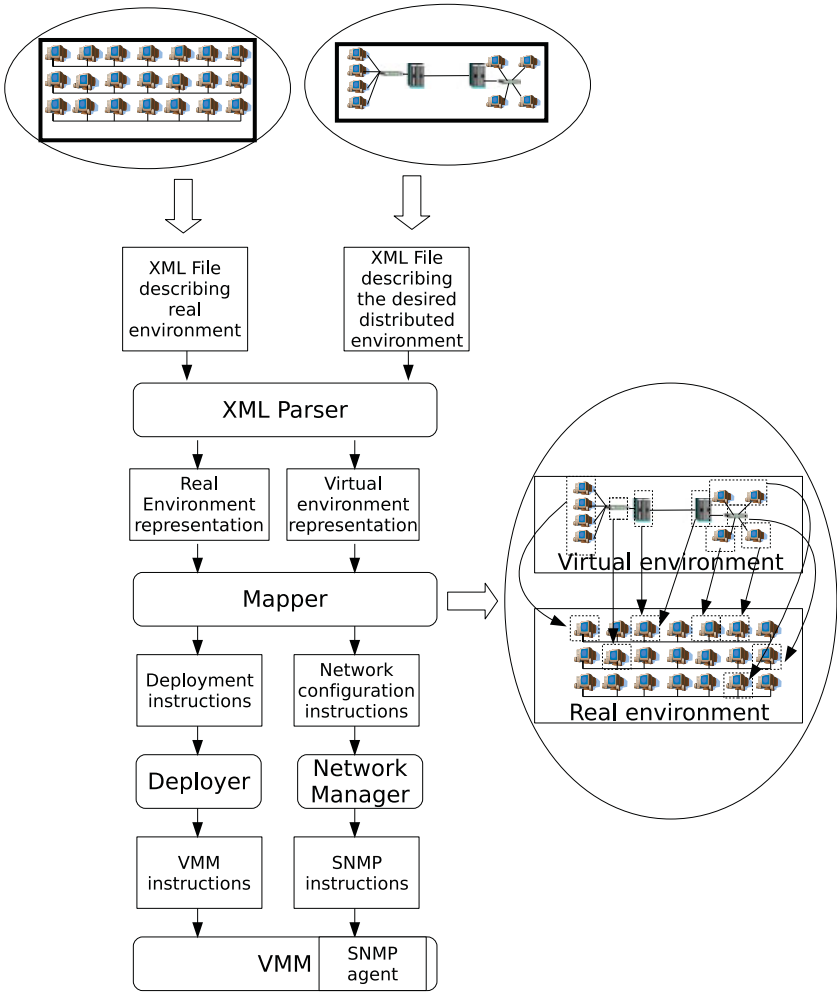

Figure 3. AEF's installation and configuration workflow.

mapping is running the experiment as quick as possible, to reduce the waiting time of other testers willing to use the environment. This limitation of one tester at a time is a design decision. Nonetheless, there is no technological limitation for tester multiplexing. If tester multiplexing is required, a mapping heuristic that minimizes the number of hosts used is preferred.

Next, the mapping problem solved by the Mapper module is detailed. Also, heuristics developed to solve such problems are presented. After AEF decides where each VM will run, the rest of

DOI: 10.1002/spe

the installation and the configuration stage, namely deployment and network configuration, start. These activities are described later in this section.

4.1. Mapping of VMs and virtual links

The Mapper module is responsible for defining the original placement of VMs and links for an experiment, as well as defining the amount of resources to be assigned to each element. Application and enforcement of such rules in the system is performed later by the other AEF modules.

The Mapper handles a cluster of workstations as a graph $c=(C, E_c)$, where C is a set of n hosts and $E_c=\{(s_i, d_i)|s_i, d_i \in C\}$ is the set of links between hosts.

The host' capacities are defined by functions $proc: C \to \mathbb{R}$, $mem: C \to \mathbb{N}$, and $stor: C \to \mathbb{R}$ that describe the processing capacity, amount of memory, and storage capacity, respectively. The processing capacity of a host may be described either in terms of a benchmark, such as SPEC CPU[‡], or may be described in terms of relative performance among the various hosts in the cluster.

The link capacity is defined by functions $bw: E_c \to \mathbb{R}$ and $lat: E_c \to \mathbb{R}$ that describe link's bandwidth and latency. For all $c_i \in C$, $bw((c_i, c_i))=\infty$ and $lat((c_i, c_i))=0$. It means that VMs running in the same host have as much bandwidth as they require to communicate, and the latency of this communication is null. This is a reasonable assumption considering that, because in this case communication is handled by the VMM itself, it is quicker and with less contention than in the case when communication goes through the physical network.

If tester multiplex is required, hosts' and links' resources availability must be updated after every tester starts its application, in order to take into account the use of resources by the already running experiments. The rest of the mapping processing, as well as the rest of the AEF installation, configuration, and execution, proceeds without modifications.

A virtual environment is a graph $v=(V, E_v)$, where V is a set of m VMs and $E_v=\{(vs_j, vd_j)|vs_j, vd_j \in V\}$ is the set of links between VMs.

The VMs' capacities are defined by functions $vproc: V \to \mathbb{R}$, $vmem: V \to \mathbb{N}$, and $vstor: V \to \mathbb{R}$ that describe the processing capacity, amount of memory, and storage capacity, respectively. The processing demand of a VM g, $proc(g)$, is described in terms of units of the expected performance of the VM comparing to a base machine.

The links' capacities are defined by functions $vbw: E_v \to \mathbb{R}$ and $vlat: E_v \to \mathbb{R}$ that describe the bandwidth and the latency, respectively.

The mapping problem consists of finding, for each $c_i \in C$, a set $G_i \subseteq V$ where the amount of resources required by all the VMs mapped to a host does not exceed the resources of the given host.

$$\bigcap_i G_i = \emptyset \quad \text{and} \quad \bigcup_i G_i = V \tag{1}$$

$$mem(c_i) \geq \sum_{g \in G_i} vmem(g) \quad \forall c_i \in C \tag{2}$$

$$stor(c_i) \geq \sum_{g \in G_i} vstor(g) \quad \forall c_i \in C \tag{3}$$

In this formulation, CPU usage is used as the variable to be optimized, and not as a constraint. This model has been chosen because (i) in the current VMMs, it is easier to control the exact amount of memory and storage than the amount of CPU; and (ii) CPU capacity is intrinsically harder to measure, define, and account for utilization. Hence, by minimizing the use of CPUs by each host we expect that the execution time of the experiment decreases. This hypothesis is validated by the experiments presented later in Section 6.

[‡]http://www.spec.org/cpu2006/.

DOI: 10.1002/spe

For the network mapping, the goal is to find, for each pair $(vs_j, vd_j) \in E_v$, a sequence $P_j = ((s_1, d_1), (s_2, d_2), \ldots, (s_p, d_p)), (s_i, d_i) \in E_c$ where

$$s_1 = c_i | vs_j \in G_i \tag{4}$$

$$d_p = c_i | vd_j \in G_i \tag{5}$$

$$s_k = d_{k-1}, \quad k = 2, \ldots, p \tag{6}$$

$$\text{for any } (s_l, d_l), (s_m, d_m) \in P_j, \quad s_l \neq s_m \quad \text{and} \quad d_l \neq d_m \tag{7}$$

$$vlat((vs_j, vd_j)) \geq \sum_{(s_k, d_k) \in P_j} lat((s_k, d_k)) \quad \forall (vs_j, vd_j) \in E_v \tag{8}$$

$$bw((s_i, d_i)) \geq \sum_{(vs_j, vd_j) | (s_i, d_i) \in P_j} vbw((vs_j, vd_j)) \quad \forall (s_i, d_i) \in E_c \tag{9}$$

Therefore, this formulation guarantees that a virtual link is mapped to a sequence of real links where: (i) the first node in the sequence is the host where the origin in the virtual link is mapped, (ii) the last host in the sequence is the host where the destination in the virtual link is mapped, (iii) there are no loops in the sequence P_j, and (iv) there are enough network resources to comply with the tester requirements.

Because in AEF model the entire cluster is available for a single tester per time, it is desirable that the execution of the experiment takes the minimum time possible. Moreover, it is undesirable that a host has a high load, because it decreases the performance of the VMs running on it, delaying the experiment. The objective function applied tries to balance the utilization of CPU on each host, considering that it can be applied in a heterogeneous environment, where hosts may have different processing powers. Thus, instead of considering the amount of VMs in each host as a load-balance metric, it uses the amount of CPU available on each host, after the mapping, as the load-balance metric. The objective function then aims at minimizing the standard deviation of the residual CPU in each host:

$$\text{minimize} \left(\sqrt{\frac{\sum_{i=1}^{n} (rproc(c_i) - \overline{rproc})^2}{n}} \right) \tag{10}$$

where

$$rproc(c_i) = proc(c_i) - \sum_{g \in G_i} vproc(g) \tag{11}$$

$$\overline{(rproc)} = \frac{\sum_{i=1}^{n} rproc(c_i)}{n} \tag{12}$$

To solve this problem, four heuristics were proposed. The initial stage is the same for all the heuristics. The links' list, which contains E_v, is sorted in descending order of bandwidth (ascending order of latency in case of ties). Thus, starting from the first element of the links' list, the unmapped VMs related to such link are chosen to be mapped. If none of the VMs was mapped, the heuristics try to map them to the same host. The criterion to select the host varies on each heuristic. If VMs do not fit a single host, the VM that requires more resources is mapped first. If the chosen host does not support the VM, the next host, according to the heuristic criterion, is tested. If no host supports the VM, the mapping fails.

The list is ordered by link bandwidth in an attempt of putting VMs with high-bandwidth links in the same host and hence saving the limited physical network bandwidth.

In all the heuristics, resources availability $ra : C \to \mathbb{R}$ of a host c_i is given by Equation (13), whereas demand $dem : V \to \mathbb{R}$ of a VM v_j, when selecting the first one to be mapped is measured

DOI: 10.1002/spe

as a normalized sum of resources availability, as shown in Equation (14).

$$ra(c_i) = \frac{mem(c_i)}{2 \times max_{c_i \in C}(mem(c_i))} + \frac{stor(c_i)}{2 \times max_{c_i \in C}(stor(c_i))} \quad (13)$$

$$dem(v_j) = \frac{vmem(v_j)}{2 \times max_{c_i \in C}(mem(c_i))} + \frac{vstor(v_j)}{2 \times max_{c_i \in C}(stor(c_i))} \quad (14)$$

The four proposed heuristics differ from each other in two points: the strategy to select the host to a given VM and whether a load-balance strategy is used after the initial mapping or not. The load-balance stage consists in changing the original placement of (migrating) VMs in order to improve the load-balance of the system, which is given by Equation (10).

To select the host to be assigned to a given VM, a list of preferential hosts is built. The elements of such a list are tested from the beginning of the list until a host with enough resources to receive the VM is found. How this list is built is different on each heuristic.

In the *LM* heuristic the host list is built in descending order of resources availability, according to Equation (13) (descending order of $proc(c_i)$ in case of ties), which means a worst-fit approach. It means that the preferred host to receive the VMs (or VM) is the least used one, i.e. the host that has more free resources. In this heuristic, after the mapping of all VMs, a migration stage takes place.

In the *LN* heuristic the host list is built in descending order of resources availability (Equation (13)), like in the previous heuristic. However, no migration takes place after the initial mapping.

In the *MN* heuristic the host list is built in ascending order of resources' availability (ascending order of $proc(c_i)$ in case of ties), which means a best-fit approach. In this heuristic, the preferred host to receive the VMs (or VM) is the one that has less availability of resources. The rationale is using the same hosts as much as possible. Also, no migration happens after the initial mapping. Although it leads to an imbalance regarding the objective function, the hypothesis is that it could be able to find valid mappings when the amount of resources required by the virtual system is close to the physical resources' availability. Considering that the rationale behind this heuristic is reducing the number of used hosts, and that application of migration would increase the number of used hosts, a version of this heuristic with migration is not evaluated.

In the *HMN* heuristic [11], the host list is built in descending order of CPU capacity ($proc(c_i)$) and descending order of resources availability (Equation (13)) in case of tie. A migration step is performed after the initial mapping to increase the system load-balance. Because the goal of this algorithm is to improve the load-balance, a version of this heuristic without migration is not evaluated.

If a heuristic applies a load-balance strategy, it consists of a migration stage whose goal is to enforce the load-balance among all hosts. At each iteration, the most loaded host is selected as the origin of the migration. The VM chosen to migrate is the one with the smallest sum of bandwidth of links to other VMs in the same host, in order to minimize the utilization of physical links. Then, starting from the least-loaded host, the load-balance factor (Equation (10)) of the environment if the migration had happened is calculated. If this value is smaller than the current load-balance factor and the chosen VM fits in the new host, the reassignment is performed. Otherwise, the next least-loaded host is considered. The process is repeated until a reassignment happens or all the hosts are tested. The whole process is repeated while the objective function improves.

After selection of hosts to each VM and eventual migration to load-balancing the hosts, it is necessary to map the links of the required virtual environment. Because several links have to be mapped, it is desirable that, after mapping each link, the bottleneck bandwidth, i.e. the smallest residual bandwidth among all the physical links, is as big as possible, to keep the possibility of mapping more virtual links over the physical links. One step toward this direction is achieved during the mapping of VMs, by mapping the ones with high communication demand in the same host. The other step toward this direction is achieved with the choice of a suitable strategy for mapping the links.

DOI: 10.1002/spe

Algorithm 1: Modified 1-constrained A*Prune.

```
Data: origin, destination, bandwidth, latency
Result: a path from origin to destination respecting bandwidth and latency constraints
for c_i ∈ C do
    ar[c_i] ← length of the Dijkstra path associated to latency from c_i to destination;
    set ← (origin,∞) (set of feasible paths and their bottleneck bandwidths);
    while set ≠ ∅ do
        bestPath ← path with the greatest bottleneck bandwidth, removed from set;
        bbw ← bottleneck bandwidth of bestPath;
        d ← last element of bestPath;
        if d=destination then
            return bestPath;
        end
        for all hosts h connected to d do
            if h ∉ bestPath then
                if bw((d,h))≥bandwidth and lat((d,h))+ar[h]≤latency then
                    set ← set ∪ (bestPath ∪ h, min(bw((d,h)), bbw));
                end
            end
        end
    end
end
```

Table II. Heuristics for mapping VMs to hosts.

Name	Hosts list sorted in ...	Load-balance migration in use?	Bandwidth reservation algorithm
LM	Descending order of capacity	Yes	A*Prune
LN	Descending order of capacity	No	A*Prune
MN	Ascending order of capacity	No	A*Prune
HMN	Descending order of CPU power	Yes	A*Prune

The strategy is the following: if the VMs are mapped to the same host, no further action is required; otherwise, the chosen strategy is the modified A*Prune [28] algorithm presented in Algorithm 1. A*Prune is an algorithm used to QoS routing in networks subject to technical constraints. In our heuristics, A*Prune has been modified to select the path with the biggest bottleneck bandwidth [23]. The distance metric for pruning inadmissible paths is the accumulated latency in the Dijkstra path between a given host and the link destination. During the pruning process, links whose available bandwidth is smaller than the required bandwidth are also pruned. Table II summarizes the four heuristics.

If at some moment a path for a virtual link cannot be found, the heuristic fails and the tester is notified about the event. If the mapping succeeds, the Deployer and the Network Manager modules use this information about placements of nodes and links to install the VMs in the chosen hosts and to configure the network routes of the VMs, respectively. This process is detailed next.

4.2. VMs deployment

After mapping the VMs to hosts, it is necessary to start execution of VMs in the selected hosts. This task is carried out by the AEF's Deployer module (Figure 4). It receives the abstract representation of the virtual environment generated by the Mapper module and uses this information to trigger the process of creating the specified VMs, with the configuration specified by the tester, in the hosts.

To make this module as independent as possible from specific deployment tools, and at the same time to allow managers to select the tools to be used, the Deployer is composed of two

DOI: 10.1002/spe

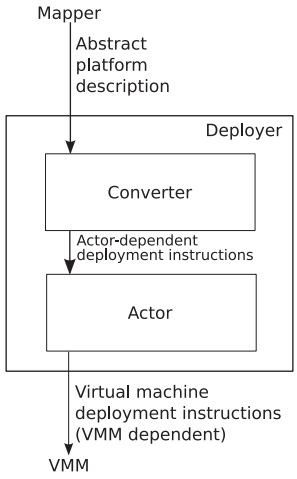

Figure 4. Deployer module.

components: the Converter and the Actor. The first one is a part of AEF and translates the internal representation of the virtual grid into the language of the specific Actor in use. The internal representation contains the machine specifications: name, amount of memory, relative amount of CPU, system image to be loaded, and MAC address (required to allow IP configuration through a virtual DHCP server, as explained in the following section).

The Actor may be either an AEF module or a script invoked by the Converter to perform the deployment. In the former case, it must be a Java package (because AEF is written in Java) to be incorporated in the AEF.

In either case, it might be necessary to transfer the VM images from the cluster frontend to the host loading the VM. These images contain a customized pre-configured operating system able to host the tester's application. Although it is not an AEF requirement, transfer and deployment of VMs are made easier with the use of technologies such as Storage Area Network (SAN) and Network Attached Storage (NAS).

The current AEF prototype is based on Xen [6] and contains three actors that deploy the machine via unicast and ssh, BitTorrent and ssh, or using WBEM management (as described in the next section), and an actor that invokes the external tool XSM [29].

It is the possibility of replacing either the Converter or the Actor that makes possible the use of other VMMs than Xen in AEF: other VMMs require the use of compatible deployment tools that can be triggered by the Actor. Other AEF modules do not handle VMs directly, and thus they are unaware of the specific virtualization technology used by the testbed.

4.3. Network configuration

After the deployment of the virtual environment, the next step is the configuration of the network. The AEF module responsible for it is the Network Manager module. It has two functions. The first one is to provide the grid behavior for the VMs taking part in the emulation. To provide such a behavior, the module provides isolation among VMs that virtually belong to different networks. This module also generates the network conditions demanded by the tester. Typically, it means to set the bandwidth and the latency between VMs 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 AEF and thus avoid the need of real DHCP and DNS servers to serve virtual nodes.

The Network Manager architecture is presented in Figure 5. It has a Converter component whose task is to translate the environment description received from the Mapper into SNMP [30] instructions. These instructions are forwarded from the SNMP Manager inside the module to

DOI: 10.1002/spe

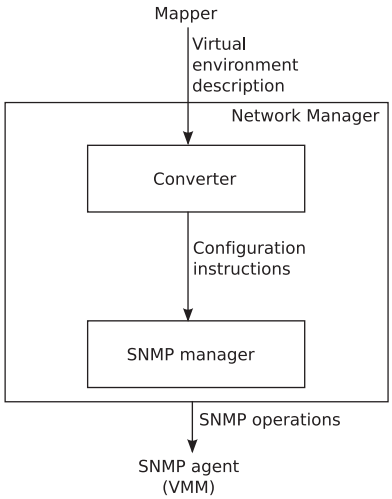

Figure 5. Network Manager module.

SNMP agents in the VMM on each node. The SNMP protocol was chosen because it is a fast and simple system management protocol, although being powerful enough to support the low-level configuration required by the module. The operations performed by this module include applying network isolation and setting network link parameters such as bandwidth and latency.

In AEF prototype the VMM used is Xen, where the configuration is applied with *iptables* configuration rules. Nevertheless, it is possible to use other VMMs in AEF. The Network Manager module would run without modifications. However, the SNMP agent must be ported to the new VMM. Porting the SNMP agent means translating each configuration operation executed by the Xen agent to the equivalent operation in the target VMM.

After execution of the network management operations, the emulated distributed system is ready to run the required experiment. The set up and the monitoring of the experiment are carried out by the Experiment Manager and its operation is discussed in the following section.

5. EXECUTION, MONITORING, AND RECONFIGURATION

With the virtual environment ready to receive the application, the next step in the AEF operation is the execution, monitoring, and reconfiguration stage. This stage starts when the applications are automatically triggered by AEF in the specified VMs. Then, the Experiment Manager Module (EMM) monitors the emulated system behavior to make sure that the usage of environment resources respects tester's specification. If not, a reconfiguration process is started to remodel the environment, changing some VMs or network parameters in order to make it comply with the original specification.

Each of the relevant actions in this stage—virtual environment management, system monitoring, and reconfiguration—is performed by a specific component of the AEF's EMM. These components and their relation with the reconfiguration cycle are presented in Figure 6 and are detailed in the rest of this section.

5.1. Virtual Environment Manager

The Virtual Environment Manager is the component from the EMM that acts directly on both the physical and the virtual environments. This component supplies services to monitor and control the life cycle of VMs—VM creation, pause/resume, configuration, and destruction. Additionally, it provides services to control applications executing inside the VMs—start and stop of applications, transfer of input files, and output retrieval.

DOI: 10.1002/spe

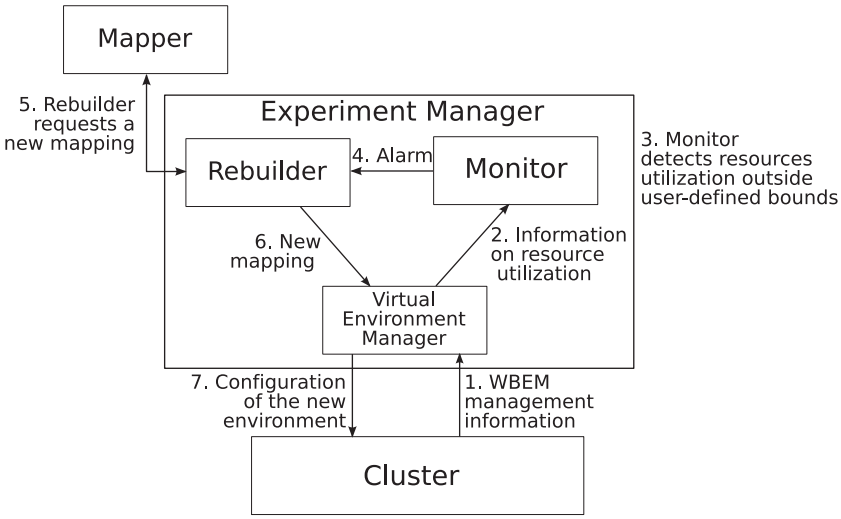

Figure 6. Experiment Manager Module components (boxes) and the role of each component in the reconfiguration process.

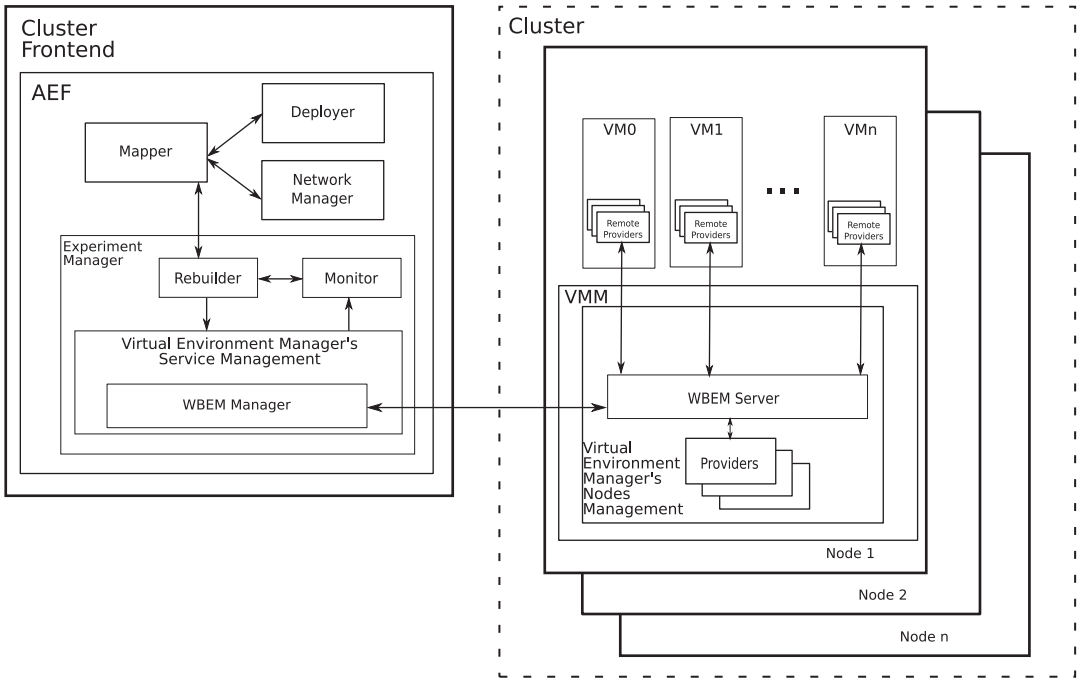

Figure 7. Virtual Environment Manager.

Figure 7 presents a detailed architecture of the Virtual Environment Manager component. It has two parts, the Service Management running in the frontend, and the Nodes Management running atop the VMM on each cluster node.

The first part of this component, the Service Management, offers a group of services to other modules from the infrastructure. This component receives requests from other components and AEF modules and invokes the Node Management located on each cluster node to perform the requested management action. The services of the Virtual Environment Manager are accessed by other components through a Java API.

DOI: 10.1002/spe

Internally, these queries correspond to services processed with the use of the WBEM specification [27]. Each service accesses WBEM manager to perform requisitions to WBEM servers present in each Node Management. Such requisitions represent either a request for any information, such as the amount of CPU used by a certain VM, or the invocation of a command, such as the creation of a VM.

The second part of this component, the Nodes Management, runs in the VMM of each host of the cluster. It is composed of a WBEM server that responds to requests from the manager, and a set of local and remote providers that manage specific features.

WBEM providers located in the VMM, together with the Nodes Management, acquire information from the machine to reply for monitoring information related to the VMM itself or to the node where the VMM runs. In the same way, providers on each VM allow monitoring of VM resource usage and also allow management of applications running in the VM. These application providers can be either specific providers for specific applications or general application providers, which receive some string representing the command to be executed and execute the command in the VM.

In our AEF prototype based on Xen, the providers run in the Xen's privileged domain and perform management operations through the Xen management API. Other methods can be used to manage VMs. In this case, only the WBEM providers are replaced by providers able to access the chosen management method. The same is required if another VMM than Xen is used in the cluster. In both cases neither the WBEM server nor the WBEM manager is replaced.

5.2. Monitor

The Monitor is the EMM component that keeps track of resource usage in both physical and virtual environments. This component has two purposes. The first purpose is to supply for tester information about usage of resources from hosts, VMs, 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 component is to generate alarms to the Rebuilder component warning it about violations in the resources usage, according to the rules defined by the tester. For example, the tester may determine that the bandwidth of a given virtual link must not exceed a specific value. In this case, if the value is exceeded, an alarm is generated and sent via a Java API to the Rebuilder component, which takes some action regarding the event.

Alarms are caused by events related to physical components of the infrastructure and by events related to virtual components of the infrastructure. A unique number that is a power of two is assigned to each different condition that triggers an alarm (e.g. CPU overutilization and link overload). The use of powers of two facilitates the identification of combined events by the Rebuilder component. These combined events are identified as the sum of the number assigned to each related event.

Each monitoring operation is translated into an operation in the Virtual Environment Manager. Thus, at a regular time interval configured beforehand, the state of each monitored resource is queried via Virtual Environment Manager. 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 with it.

After all the relevant information is obtained, the elements that caused similar alarms are grouped together to generate a single alarm related to that specific violation. Subsequently, each generated alarm is passed to the Rebuilder component.

5.3. Rebuilder

The function of the Rebuilder component is to reconfigure the environment after detection of a violation in the resources' utilization. This component determines the characteristics of the new virtual environment and verifies that if such new virtual environment can be mapped to the real environment.

To handle this task, this component receives the alarms from the Monitor, and, considering the specific alarm, defines the action to be taken.

DOI: 10.1002/spe

The set of actions that the Rebuilder chooses from in response to an alarm are either simple actions, caused by single alarms, or complex actions, caused by the activation of more than one alarm. This information is received from the Monitor as an event number. For each event number, there is a list of actions to be taken. These actions are composed of operations under the virtual environment: creation and/or destruction of VMs, change in the VM parameters, or change in the virtual network parameters. If some action requires changing the amount of VMs in the sites, it is done according to tester's instructions. Hence, some actions cannot be applied because they might violate the minimum or the maximum amount of VMs specified by the tester.

After the actions are processed, the parameters of the new system are defined by the Rebuilder. The new virtual environment configuration is then sent to the Mapper to try to map the new environment in the cluster. If a new mapping is not found, the Rebuilder tries to apply the next action from the list in the environment and the process is repeated. Actions that would cause the system to return to a state already tested are removed from the actions' list of the new configuration.

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 detected during experiment execution is generated to the tester together with the regular experiment output.

When the experiment runs from the beginning till the end without violations in the usage of resources, only the regular experiment output and the execution logs are generated. If no further actions are required, the cluster is cleared (VMs are destroyed) by the Virtual Environment Manager, virtual DHCP and DNS servers are stopped and the AEF becomes available to the next tester.

6. EVALUATION

This section presents evaluations of each AEF component. It starts with the evaluation and the comparison of the mapping heuristics and the rationale behind them. Next, experiments running in AEF are validated through a comparison with its execution in a real environment. Finally, execution, monitoring, and reconfiguration stages are evaluated.

The experiments aimed at validating AEF components were based on an AEF prototype we implemented. The mapping heuristics, on the other hand, were evaluated with the use of discrete-event simulation, in order to allow investigation of their behavior in different scenarios.

6.1. Evaluation and comparison of mapping heuristics

The goal of the tests presented in this section is to evaluate the four heuristics presented in Section 4.

6.1.1. Evaluation scenario. The four mapping heuristics presented in Section 4 were evaluated using discrete-event simulation. The CloudSim [31] simulation toolkit is used in the tests. The simulation scenarios are created by a generator that receives as input number of hosts, number of VMs, and virtual network density. The generator uses this information to create the links between VMs and to assign a given amount of resources to each one. The amount of resources (memory, storage, CPU) are randomly generated, based on a uniform distribution.

The physical environment of the simulation scenario is composed of two different cluster topologies. The first one is a 2-D torus topology. The second cluster topology is a switched topology, in which hosts are connected to cascaded 64-port switches. In both cases, connections between hosts (or connections between a host and a switch) have 1 GB of bandwidth and 5 ms of latency. Both cluster topologies are built with the same set of hosts. To represent heterogeneity in the cluster, resources of each of the 40 hosts in the cluster are randomly generated, using the generator described in the previous paragraph. Host memory varies uniformly between 1 and 3 GB. Storage varies between 1 and 3 TB and CPU capacity varies between 1000 and 3000 MIPS. The former are the units used by CloudSim to model CPU capacity, and it can be used either to supply

DOI: 10.1002/spe

Table III. Summary of simulation setup.

	Physical environment	Virtual environment	
		Low-level applications	High-level applications
Topology	2-D torus, switched	Graph, density 2.5%	Graph, density 2.5%
Bandwidth	1 GBps	87–175 kbps	0.5–1 Mbps
Latency	5 ms	30–60 ms	30–60 ms
Nodes	40	100–1000	100–500
Memory	1–3 GB	19–38 MB	128–256 MB
Storage	1–3 TB	19–38 GB	100–200 GB
CPU	1000–3000 MIPS	19–38 MIPS	50–100 MIPS

a relative performance value for the tasks (this is the approach used in this work) or to model real machines that were previously evaluated by a benchmark.

Regarding the workloads, two different use cases are considered: a virtual environment for high-level applications and a virtual environment for low-level applications. The first case represents emulation of applications, such as grid computing applications, cloud computing middleware, and other cases, where the application runs in a system containing the operating system, the application, libraries, and supporting software for applications (e.g. a database management system or a Java VM), which demand VMs with large amount of memory and storage. This scenario resembles emulation experiments like the ones presented in the following section.

The second case considers an environment where the VMs run, for example P2P protocols. In this case, VMs do not need to run applications that require many resources. Instead, smaller VMs that run only the basic software are used. Thus, in this case the VMs require less memory and storage. This scenario is based on emulation experiments presented by Quétier *et al.* [14]. Table III summarizes the experimental setup.

In the high-level applications' workload, the number of VMs in each simulation varies between 100 and 500. Memory of each VM varies uniformly between 128 and 256 MB. Storage of each VM is uniformly distributed between 100 and 200 GB. The MIPS required by each VM varies uniformly between 50 and 100 MIPS. Links between VMs have bandwidth randomly defined, with its value between 0.5 and 1 Mbps and latency uniformly distributed between 30 and 60 ms.

In the low-level applications' workload, the number of VMs in each simulation varies between 100 and 1000. Memory of each VM varies uniformly between 19 and 38 MB. Storage of each VM is uniformly distributed between 19 and 38 GB. The MIPS required by each VM varies uniformly between 19 and 38 MIPS. Links between VMs have bandwidth randomly defined, with its value between 87 and 175 kbps and the latency uniformly distributed between 30 and 60 ms.

In both workloads, links between VMs are randomly set. The number of links created is determined by the graph density, which is 2.5%. The algorithm used to generate the graph topology guaranteed that the output graph is connected.

Each scenario is simulated with each workload 50 times. Each time, a new cluster and a new distributed system are randomly generated, according to the scenario parameters. The average value of each output—mapping time, simulation time, and objective function—are collected. The results are presented as follows.

6.1.2. Heuristics comparison. Figure 8 depicts the average objective function (Equation (10)) observed for each heuristic in each scenario. The scenarios are described by the number of VMs.

In scenarios with less VMs, there are more opportunities for load-balancing migrations, and the migrations contribute for improvements in the objective function of HMN and LM. Because LN and MN do not have a load-balancing stage, the objective function for these heuristics is higher than for the other heuristics. Furthermore, because MN minimizes the number of hosts used, the imbalance, and consequently the objective function, is higher regardless of the number of machines used. Nevertheless, MN was the only heuristic able to map 500 machines in the high-level scenario.

DOI: 10.1002/spe

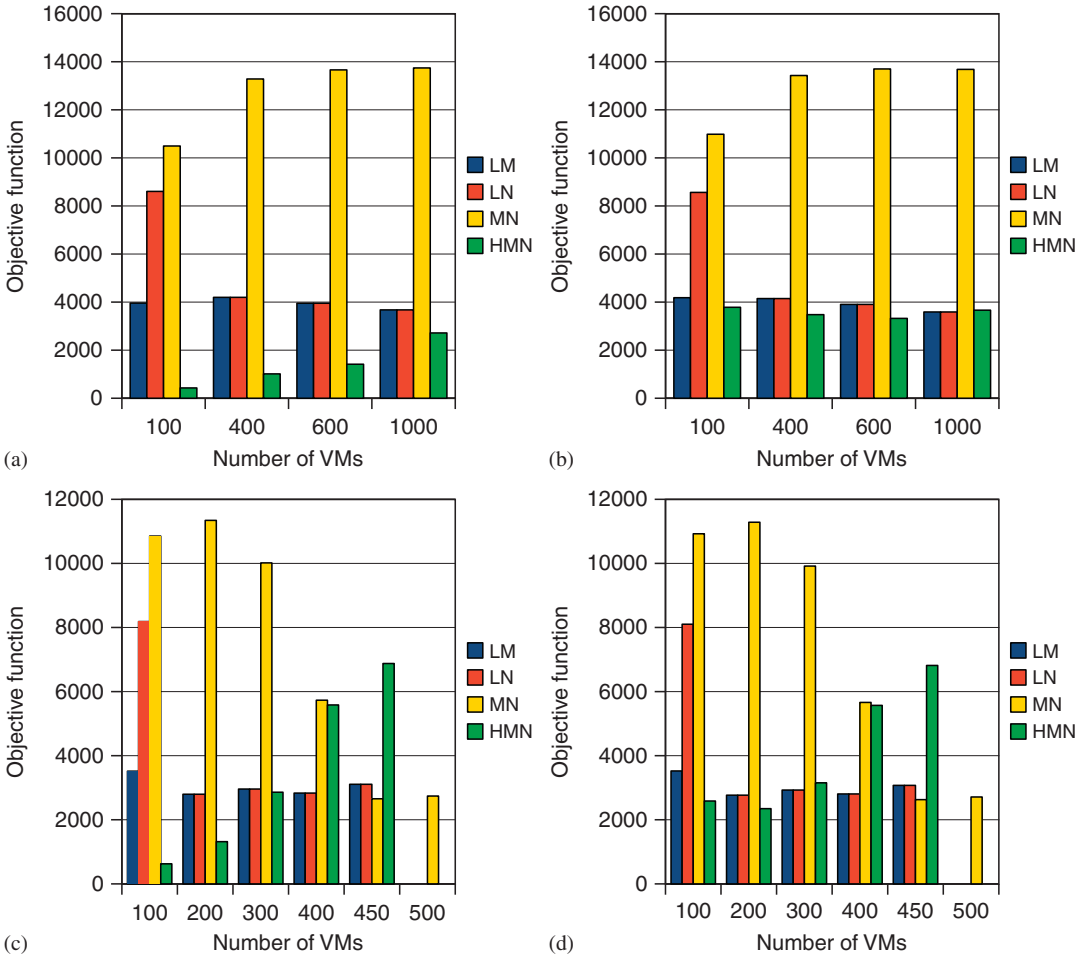

Figure 8. Objective function in different simulation scenarios: (a) low-level workload, torus cluster; (b) low-level workload, switched cluster; (c) high-level workload, torus cluster; and (d) high-level workload, switched cluster.

When the number of VMs is high, there are fewer opportunities for migrations and then the performances of LM and LN are equivalent, both in terms of the use of hosts and in the objective function. It happens because the heuristics tend to finish the mapping with almost the same configuration, as the initial mapping is the same for both heuristics and only a few migrations happen in this case.

The different approach for selecting the hosts in the HMN heuristics leads to better mappings when the number of hosts is small. However, in the presence of a bigger number of VMs, and fewer opportunities for migrations, the initial choice of VMs performed by HMN leads to worse mappings.

Figure 9 shows the mapping time in each scenario. The time showed was obtained in a Pentium 4 2.8 GHz with 1 MB of cache and 2560 MB of RAM memory running Linux Debian Etch. The mapping time is dominated by the time to execute the A*Prune algorithm and, ultimately, by the determination of the shortest path for each virtual link. In the torus topology, the number of possible paths is bigger and then it requires more time compared to the same scenarios with a switched topology.

The high amount of time required by MN in the torus topology makes it unsuitable for utilization in some practical scenarios. Nevertheless, MN outperforms the other heuristics in the switched cluster, and hence it may be used in such topologies. LM is slower than LN because of the

DOI: 10.1002/spe

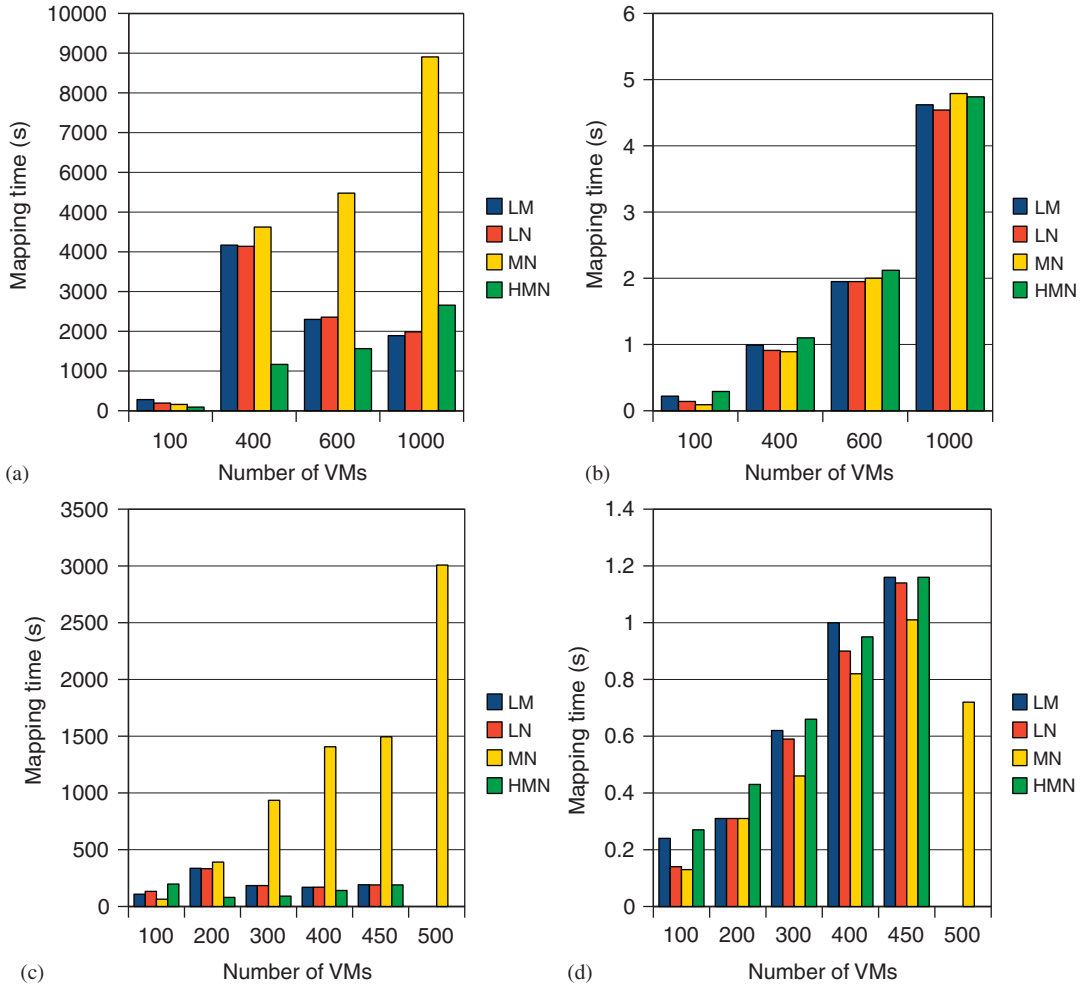

Figure 9. Mapping time in different simulation scenarios: (a) low-level workload, torus cluster; (b) low-level workload, switched cluster; (c) high-level workload, torus cluster; and (d) high-level workload, switched cluster.

migrations. HMN performs better than the other heuristics in most experiments in the torus cluster. However, it is worse than LM and LN in some scenarios.

Mapping 1000 VMs (12 487 virtual links) in a torus topology with 80 physical links requires up to half an hour for LM and LN, two and a half hours in MN, and 45 min in HMN. In the switched topology, as there is only one possible mapping for each virtual link (from the first host to the cascaded switches until reaching the one where the second host is connected and then to the second host), the mapping time is less than 5 s in all scenarios.

The mapping of VMs to hosts determines the mapping time. This is because, when two VMs are mapped to the same host, the link between them does not have to be mapped using the A*Prune algorithm. Furthermore, the use of the available bandwidth by the links previously mapped gradually reduces the search space of the A*Prune algorithm when virtual links are successively mapped. Thus, the more the links are used, the less they are likely to be used further, and with a smaller solution space to be covered, the execution time of A*Prune reduces, and so does the overall mapping time. This is the reason why the mapping time reaches a peak at 400 VMs in the low-level workload and 200 VMs in the high-level scenario and then becomes smaller for greater number of VMs. This is another factor for the poor performance of MN: as more VMs are mapped to the same host in this heuristic, the search space for the A*Prune is reduced to a smaller rate than in other strategies.

DOI: 10.1002/spe

Concluding, no single heuristic is better than the others in all scenarios. Hence, some criteria must be applied to choose the fittest heuristic for each instance of the mapping problem. If the amount of resources required by the virtual system is close to the amount of the available resources, MN is the best heuristic. For a small number of VMs, HMN provides a better, although slower, mapping than the other heuristics. However, HMN mapping is not stable, in the sense that when the number of VMs increases the mapping looses quality, whereas LM keeps a nearly constant mapping quality. Hence, if the rate of VM/host is lower, HMN is a good choice. Otherwise, LM is the preferred heuristic. LN performs always worse or equal to LM, and it is only slightly faster than LM. Hence it is not worth applying such heuristic.

*6.1.3. Effectiveness of the modified A*prune algorithm.* To evaluate the importance of the A*Prune algorithm in the mapping process, a modified HMN heuristic was developed. The modified version does not use A*Prune to map the virtual links. Instead, a depth-first search algorithm is used to map virtual links to paths. Because the initial mapping is the same, the objective function is the same in both strategies. Nevertheless, the modified heuristic failed in finding a valid mapping in 54.31% of the scenarios where the heuristic with A*Prune found valid mappings. Moreover, in none of the simulated scenarios the modified heuristic succeeded to find a mapping when the A*Prune heuristic failed. It shows that the application of the modified A*Prune contributes to the success of the mapping of virtual environments to physical environments.

6.1.4. Effectiveness of the objective function. Regarding our initial hypothesis that the chosen objective function (Equation (10)) reduces the experiment execution time, there is a correlation of 0.7 between the objective function and the execution time of the experiment in the simulated environment. It supports the use of Equation (10) as a suitable representation of an objective function for a mapping aiming at reducing the execution time of an emulation.

6.2. AEF evaluation

In this section, we present an evaluation of AEF, by comparing the results of a real-world experiment presented elsewhere [32] with the results of the same experiment in an emulation environment built and managed by AEF. The goal of the comparison is to validate the mapping and the execution components of AEF, mainly regarding network connections and bandwidth limitations. Thus, we are not concerned in this paper with the design and the implications of the experiment results in the context it was originally applied.

The experiment evaluates the makespan of jobs running in an OurGrid [33] grid using the SRS scheduler [32]. The environment used for the real-world test is composed of 50 machines in two OurGrid sites located at 4000 km apart. One site is used as a resource consumer, and the other is used as a resource provider. The resource provider hosts a cluster whose machines are opportunistically delivered to the grid. The supplier has 48 grid machines plus one OurGrid peer. The machines are 11 Pentium 3 1.0 GHz with 256 MB of RAM memory, 10 Pentium 4 1.6 GHz with 256 MB of memory, 9 Dual Pentium 3 550 MHz with 256 MB of memory, 4 Dual Pentium 3 1 GHz with 256 MB of RAM memory, 8 Pentium 4 2.8 GHz with 2.5 GB of RAM memory, and 6 Dual Xeon 3.6 GHz with 2 GB of memory. Only one CPU of the dual machines is used. The consumer site has only one machine, which contains both the peer and the MyGrid scheduler (both are part of the OurGrid system).

The grid job executed in both experiments contains 12 tasks. Each one sends a file, executes a sleep call of 5 min, and receives a file of the same size as that of the file sent. The job is executed four times: the first one without file transferring and the others with different file sizes: 100 kB, 1 MB, and 10 MB. To simulate the dynamism of a grid environment, resources are randomly removed from the grid at every 10 min.

The network parameters used in the virtual network were obtained with the observation of the bandwidth obtained in a data transfer between the two actual sites using *scp* (for the bandwidth) and with the latency measured by the *hping2* tool, which are, respectively, 2 Mbps and 200 ms. Each node in the virtual network has 256 MB of RAM memory and 1 GB of storage. Although

DOI: 10.1002/spe

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

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

some machines from the real environment have more memory than the machines in the emulated environment, it does not compromise the experiment, because the application uses less than 128 MB of memory.

The cluster used to host the emulated environment is composed of eight Pentium 4 2.8 GHz with 1 MB of cache and 2560 MB of RAM memory. Cluster machines are connected by a dedicated Fast Ethernet switch. Machines run Xen VMM 3.1, and the Xen's privileged domain uses 328 MB of the available RAM memory. Thus, 2232 MB were available to the VMs on each host. No network traffic but the one generated by this experiment is present in the physical environment during the tests.

AEF built, in the installation and configuration stage, the virtual environment using the whole cluster. Deployment happened with an XSM [29] compatible module. The *opmanager*[§] tool was used to assure that the sites were isolated from each other and behaved according to the specified configuration.

Table IV presents the observed makespan of the job in both real and virtual environments. It also shows the deviation, i.e. 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 the emulated results is less than 10% for all the cases. However, this value increases with the size of the file being transferred. It happens because of 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 networks becomes smaller. However, the emulated network is faster than the real network. Hence, when larger files are transferred, the difference between the emulated and the real networks causes a bigger influence in the results of the experiment. The causes of the deviation in the behavior of emulated and real networks are the following:

Error in the acquisition of network parameters: Measurement of network parameters is a difficult task, particularly when it involves machines belonging to several administrative domains, in which common methods to evaluate it are blocked by systems administrators. To circumvent it, tools running in the application layer were used. Hence, an inaccuracy in the values used to set the emulated environment is expected. This error may be reduced by using more accurate methods to acquire bandwidth and latency values.

Fluctuation in the real network traffic: Network parameters are not static, and variation in the network traffic led to variation in the available latency and bandwidth. Hence, their values vary during the execution of the real experiment. During emulation, however, these parameters are statically defined. Fluctuation in network values can be injected in the experiment in the following way. The tester 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. In this way, it is possible to decrease the deviation between real and emulated experiments.

Error in the network emulation: There are differences between the configured network parameters and the values obtained in the emulated network. Also, I/O isolation in current Xen implementation is not as effective as CPU isolation. Hence, for I/O-bound applications, performance degrades if the I/O rate is too high. The mapping heuristics help in avoiding this effect by scattering VMs among hosts. Because the heuristics try to use as many hosts as possible, the number of VMs per

[§] http://www.opmanager.com/.

DOI: 10.1002/spe

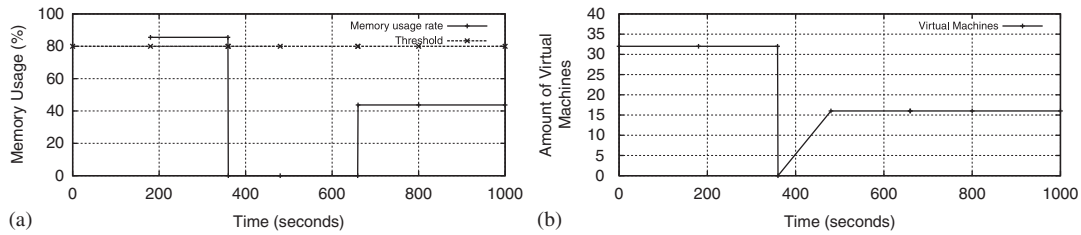

Figure 10. (a) Memory utilization of one virtual machine and (b) number of virtual machines in the experiment.

host tends to decrease and so does the interference caused by I/O requisitions from VMs running in the same host.

Although all those factors interfere with the results, they are close to the ones observed in the real experiment. As the results in Table IV suggest, the error tends to accumulate in time. Hence, short-time experiments tend to have better results than the long-time experiments. Nevertheless, even an experiment that took almost 3 h to complete in a real environment presented a deviation below 10%, which shows that AEF is able to successfully emulate grid systems.

6.3. Execution, monitoring, and reconfiguration evaluation

In this section, we present an evaluation of AEF's EMM. The goal of this experiment is to verify that if this module is able to detect and react accordingly to violations in resource usage by applications. If necessary, the EMM should trigger a new mapping and deploy a new environment that complies with the tester specification. Four machines of the cluster used to host the emulated environment described in the previous section are used in this experiment.

The experiment input is a partial description of a local area network. The initial configuration proposed by the tester is composed of 32 VMs with 256 MB of memory each. According to the input rules, the system can be scaled down to two VMs, and can be scaled up without restriction, as long as the use of memory on each VM is kept below 80%. Since each physical node has a limited memory capacity, to increase the memory capacity of individual VMs mapped to this node AEF has to reduce the number of VMs per host.

To force the alarm mechanism to be activated, the experiment description includes instructions to run an application that allocates 200 MB of RAM memory on each VM. With this allocation request, it is assured that memory utilization on VMs is above the threshold (in this case 80%) and consequently the alarm and reconfiguration mechanism will be activated. Additionally, this experiment validates the mechanism for automatic triggering of applications inside VMs.

The experiment description and the virtual environment description are supplied to AEF that uses this information to build the virtual environment using the regular installation and configuration stages described in Section 4. Then, the EMM executes the experiment.

Figure 10(a) shows the memory utilization of one VM during the experiment and Figure 10(b) shows the number of VMs running in the experiment. Initially, there is no information about resource usage because this data has not been collected in any of the running VMs. After 3 min (as defined by the tester in this experiment), information about memory utilization is collected. The Monitor, one of the three EMM components (Figure 6), detects that memory utilization in the VMs is above the configured threshold of 80% and triggers an alarm that activates the Rebuilder. The first reconfiguration action considered (doubling the amount of memory on each VM) fails because there is not enough physical hosts to accommodate 32 VMs with 512 MB of RAM. The second action, doubling the memory and reducing the number of VMs from 32 to 16, allows the Mapper to find a valid mapping. In the new mapping, four VMs are allocated to each host. Because a valid mapping is found, reconfiguration procedures are triggered in the system.

After the reconfiguration of the environment that finished at $t = 480$ s, the application used in this experiment is restarted. With this new configuration, the memory usage in the VMs is below

DOI: 10.1002/spe

the defined threshold and the experiment finishes without further reconfigurations, which is the expected behavior of the system.

7. CONCLUSION AND FURTHER WORK

We presented in this paper the AEF, a virtualization-based testbed for grid applications. AEF implements a full automated emulation of a grid infrastructure in a cluster of workstations enabling controlled execution of experiments. The development of such testbed involved combined efforts in three areas; virtualization, capacity mapping, and system management.

In the virtualization area, we contributed with mechanisms for the automatic management of both the physical and the emulated infrastructures. In the area of capacity mapping, we introduced a formal definition of the mapping problem and described the mechanisms to allow automatic mapping of VMs to hosts, VMs deployment, and network configuration. Four heuristics to solve the problem were presented and compared in different scenarios. In the area of system management, we described and validated an architecture to allow not only automatic control of applications in the emulated environment but also system monitoring and reconfiguration.

With the presented experiments, we showed that AEF is an excellent alternative for the emulation of distributed systems because it uses regular and widely available clusters of workstations without requiring any special hardware to operate. It also allows a high level of control and monitoring over the executed application that would not be possible in real environments. Although AEF uses virtualization techniques, it allows testers to abstract all the aspects of the operation and management of virtualization tools and environments, which is a major difference to similar approaches found in the literature.

Some AEF components can be used in other areas besides emulation. For example, deployment, network configuration, and system monitoring are regular topics concerning large data centers. Because of that, all the techniques presented in this paper to manage VMs in the context of emulation can be applied in such environments. The mapping heuristics described in this paper can also be applied in Cloud computing [4] environments. Clouds typically require the use of large amounts of VMs in data centers, the same scenario addressed by our mapping problem.

Also regarding the mapping problem, we intend to investigate the impact of other metrics of CPU utilization in the model. For example, the impact of an objective function based on the ratio of CPU usage instead of the amount of remaining CPU will be the focus of future research. Another point to be investigated in the future is the development of an abstraction layer, between the tester and the AEF, to allow testers without knowledge in distributed systems to use AEF to evaluate their distributed application without having to describe the distributed environment.

ACKNOWLEDGEMENTS

Part of this paper gathers, expands, and updates the previous results published in three conference papers: [9] for the installation and configuration stages, [10] for the execution, monitoring, and reconfiguration stages, and [11] for the modeling presented in Section 4.1 and the HMN heuristic. This work is partially supported by the CAPES PDEE research grant 1185-08-0, the Australian Research Council (ARC), and the Department of Innovation, Industry, Science and Research (DIISR). This work was primarily carried out at CLOUDS Lab during Rodrigo Calheiros's visit to the University of Melbourne.

REFERENCES

1. Foster I, Kesselman C (eds.). *The Grid*: *Blueprint for a New Computing Infrastructure*. Morgan Kaufmann: San Francisco, U.S.A., 1999.
2. Sulistio A, Yeo CS, Buyya R. A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. *Software*: *Practice and Experience* 2004; **34**(7):653–673.
3. Quétier B, Cappello F. A survey of grid research tools: Simulators, emulators and real life platforms. *Seventeenth IMACS World Congress on Scientific Computation*, *Applied Mathematics and Simulation*. IMACS: Paris, France, 2005.

DOI: 10.1002/spe

4. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems* 2009; **25**(6):599–616.
5. Hibler M, Ricci R, Stoller L, Duerig J, Guruprasad S, Stack T, Webb K, Lepreau J. Feedback-directed virtualization techniques for scalable network experimentation. *Technical Note FTN-2004-02*, University of Utah Flux Group, 2004. Available at: http://www.cs.utah.edu/flux/papers/virt-ftn2004-02.pdf [25 July 2008].
6. Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. Xen and the art of virtualization. *Nineteenth Symposium on Operating Systems Principles*. ACM: Bolton Landing, U.S.A., 2003; 164–177.
7. Devine SW, Bugnion E, Rosenblum M. Virtualization system including a virtual machine monitor for a computer with a segmented architecture. *US Patent 6397242*, 2002.
8. Smith JE, Nair R. *Virtual Machines*: *Versatile Platforms for Systems and Processes*. Morgan Kauffmann: San Francisco, U.S.A., 2005.
9. Calheiros RN, Storch M, Alexandre E, De Rose CAF, Breda M. Applying virtualization and system management in a cluster to implement an automated emulation testbed for grid applications. *Twentieth International Symposium on Computer Architecture and High Performance Computing*. IEEE Computer Society: Campo Grande, Brazil, 2008; 97–104.
10. Calheiros RN, Alexandre E, do Carmo AB, De Rose CAF, Buyya R. Towards self-managed adaptive emulation of grid environments. *IEEE Symposium on Computers and Communications*. IEEE: Sousse, Tunisia, 2009; 818–823.
11. Calheiros RN, Buyya R, De Rose CAF. A heuristic for mapping virtual machines and links in emulation testbeds. *Thirty-Eighth International Conference on Parallel Processing*. IEEE Computer Society: Vienna, Austria, 2009; 518–525.
12. Kamp PH, Watson RNM. Jails: Confining the omnipotent root. *Second International System Administration and Networking Conference*. SANE: Maastricht, Netherlands, 2000. Available at: http://www.sane.nl/events/sane2000/papers/kamp.pdf.
13. Jiang X, Xu D. vBET: a VM-based emulation testbed. *ACM SIGCOMM Workshop on Models*, *Methods and Tools for Reproducible Network Research*. ACM: Karlsruhe, Germany, 2003; 95–104.
14. Quétier B, Jan M, Cappello F. One step further in large-scale evaluations: The V-DS environment. *Research Report RR-6365*, Institut National de Recherche en Informatique et en Automatique, 2007.
15. Canonico R, Gennaro PD, Manetti V, Ventre G. Network emulation on Globus-based grids: Mechanisms and challenges. *Grid Enabled Remote Instrumentation*, Davoli F *et al*. (eds.). Springer: Berlin, 2009; 455–468.
16. Apostolopoulos G, Hassapis C. V-eM: A cluster of virtual machines for robust, detailed, and high-performance network emulation. *Fourteenth International Symposium on Modeling*, *Analysis*, *and Simulation of Computer and Telecommunication Systems*. IEEE Computer Society: Monterey, U.S.A., 2006; 117–126.
17. Childs S, Coghlan B, Walsh J, O'Callaghan D, Quigley G, Kenny E. A virtual TestGrid, or how to replicate a national grid. *Proceedings of the EXPGRID Workshop at the Fifteenth International Symposium on High Performance Distributed Computing*. IEEE Computer Society: Paris, France, 2006.
18. Childs S, Coghlan B, McCandless J. GridBuilder: A tool for creating virtual grid testbeds. *Second IEEE International Conference on e-Science and Grid Computing*. IEEE Computer Society: Amsterdam, Netherlands, 2006; 77–84.
19. Gupta D, Vishwanath KV, Vahdat A. DieCast: Testing distributed systems with an accurate scale model. *Fifth USENIX Symposium on Networked Systems Design and Implementation*. USENIX: San Francisco, U.S.A., 2008; 407–421.
20. Singh A, Korupolu M, Mohapatra D. Server-storage virtualization: Integration and load balancing in data centers. *ACM/IEEE Conference on Supercomputing*. IEEE Computer Society: Austin, U.S.A., 2008.
21. Khanna G, Beaty K, Kar G, Kochut A. Application performance management in virtualized server environments. *Tenth Network Operations and Management Symposium*. IEEE Computer Society: Vancouver, Canada, 2006; 373–381.
22. McNett M, Gupta D, Vahdat A, Voelker GM. Usher: An extensible framework for managing clusters of virtual machines. *Twenty-First Large Installation System Administration Conference*. USENIX: Dallas, U.S.A., 2007; 167–181.
23. Sundararaj AI, Sanghi M, Lange JR, Dinda PA. An optimization problem in adaptive virtual environments. *ACM SIGMETRICS Performance Evaluation Review* 2005; **33**(2):6–8.
24. Ricci R, Alfeld C, Lepreau J. A solver for the network testbed mapping problem. *ACM SIGCOMM Computer Communication Review* 2003; **33**(2):65–81.
25. Liu Y, Li Y, Xiao K, Cui H. Mapping resources for network emulation with heuristic and genetic algorithms. *Sixth International Conference on Parallel and Distributed Computing*, *Applications and Technologies*. IEEE Computer Society: Dalian, China, 2005; 670–674.
26. Legrand A, Marchal L, Casanova H. Scheduling distributed applications: The SimGrid simulation framework. *Third International Symposium on Cluster Computing and the Grid*. IEEE Computer Society: Tokyo, Japan, 2003; 138–145.
27. Harnedy S. *Web-based Information Management*: *An Introduction to the Technology and Its Application*. Prentice-Hall: Upper Saddle River, U.S.A., 1998.
28. Liu G, Ramakrishnan KG. A* Prune: An algorithm for finding K shortest paths subject to multiple constraints. *Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*. IEEE: Anchorage, U.S.A., 2001; 743–749.

DOI: 10.1002/spe

29. Franciosi F, Orengo JP, Storch M, Grazziotin F, Ferreto T, De Rose CAF. Deploying and managing Xen sites with XSM. *Workshop on Virtualization/Xen in HPC Cluster and Grid Computing Environments*. Springer: Rennes, France, 2007; 195–204.
30. Stallings W. *SNMP*, *SNMPv2*, *SNMPv3*, *and RMON 1 and 2* (3rd edn). Addison-Wesley: Reading, U.S.A., 1999.
31. Buyya R, Ranjan R, Calheiros RN. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. *Seventh High Performance Computing and Simulation*. IEEE: Leipzig, Germany, 2009; 1–11.
32. Calheiros RN, Ferreto T, De Rose CAF. Scheduling and management of virtual resources in grid sites: The site resource scheduler. *Parallel Processing Letters* 2009; **19**(1):3–18.
33. Cirne W, Brasileiro F, Andrade N, Costa LB, Andrade A. Labs of the world, unite!!!. *Journal of Grid Computing* 2006; **4**(3):225–246.

DOI: 10.1002/spe