A Heuristic for Mapping Virtual Machines and Links in Emulation Testbeds

Rodrigo N. Calheiros[1,2], Rajkumar Buyya[2], César A. F. De Rose[1]
[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@pucrs.br, raj@csse.unimelb.edu.au, cesar.derose@pucrs.br

Abstract

Distributed system emulators provide a paramount platform for testing of network protocols and distributed applications in clusters and networks of workstations. However, to allow testers to benefit from these systems, it is necessary an efficient and automatic mapping of hundreds, or even thousands, of virtual nodes to physical hosts—and the mapping of the virtual links between guests to physical paths in the physical environment. In this paper we present a heuristic to map both virtual machines to hosts and virtual links between virtual machines to paths in the real system. We define the problem we are addressing, present the solution for it and evaluate it in different usage scenarios.

1. Introduction

Testing of applications, middleware, and protocols for platforms such as grid computing [7], P2P computing [10], utility computing [12], and cloud computing [3] is a very challenging task. That is because the hardness in getting access to third-parties resources in order to enforce specific environment conditions and to enable test replication. Another problem is to enforce specific network conditions during the test. Because tests can run for a long time, variation in network conditions is unavoidable in real environments, and it is beyond the control of the tester.

To overcome such limitations, several distributed system emulators built upon virtualization technology were proposed [1, 4–6, 11]. These projects aim at delivering a scalable and controlled testbed where a tester can evaluate the behavior of the actual system. Because emulators allow testers to describe the exact configuration and conditions of the emulated environment, it is also possible not only to reuse a given emulated environment but also to reproduce tests. Moreover, these platforms benefit from virtualization technology, which provides an easy way to multiplex emulation nodes.

One aspect that must be considered in virtualization-based emulators is the placement of virtual machines in the hosts, because it limits the scalability of the environment due to fragmentation problems. For example, even if the overall amount of free memory in the testbed allows more virtual machines to be deployed in the environment, it is possible that none of the hosts has enough memory to support a new virtual machine. The same may happen to other host resources (e.g., CPU, storage).

Nevertheless, finding a mapping of virtual machines to hosts solves part of the problem of mapping a virtual environment to physical environments. Another problem is finding, for each link between two virtual machines, a path in the physical infrastructure going from the host that runs one of the virtual machines to the host running the other one. Depending on the cluster topology and hosts' configuration, the hosts where the virtual machines are running may not be directly connected, hence the path passes by other hosts. Furthermore, as in the case of mapping virtual machines, network resource constraints must be considered during the map of the links.

In this paper, we present a heuristic to map virtual machines and virtual links to physical machines and network paths in the real system. The heuristic goal is to optimize the utilization of the physical environment by balancing the load among physical machines that may be heterogeneous, i.e., that may have different amounts of memory, storage, and CPU power.

The work presented in this paper is part of a project aiming at developing a fully-automated virtualization-based emulator [4] for distributed systems. The emulator is able to build the virtual system and trigger the applications. The automatic mapping of virtual machines and links is an important step of the process of building the emulated environment.

2. Related Work

There are several projects aiming at using virtualization technology to develop distributed systems emulators. However, the approach used to map virtual machines to hosts, if any, is different on each one.

In both V-DS [11] and TestGrid [6] systems, the mapping is performed manually by the tester. The problem of manually mapping is that it becomes a laborious and error-prone work when the number of virtual machines to be mapped reaches hundreds or thousands.

V-eM [1] handles the problem of mapping the virtual machines to the hosts considering that the hosts are connected by a switch and that every virtual link corresponds to one real link between network interfaces. Although switches are widely used for connection of cluster nodes, there are other topologies that V-eM cannot handle, for example, clusters with torus or ring topology. Finally, this approach does not allow the mapping of virtual links between guests whose hosts are not connected in the same switch.

NEPTUNE [5] considers the same topology than V-DS. However, no specific solution is proposed for the mapping problem.

Differently from all these systems, our approach can manage arbitrary cluster networks. Consequently, it can be applied in any one of the previous approaches to improve the mapping.

The mapping problem addressed in this paper has similarities with problems found in other contexts. *Generic Adaptation Problem in Virtual Execution Environments* (GAPVEE) [17] consists in finding a mapping of virtual machines to hosts and the mapping of virtual links to paths in order to emulate a local area network in a wide area network, while in our work we address the opposite problem, that is, emulating 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 our formulation, on the other hand, the goal is to find a mapping starting from a state where there are no virtual machines mapped. Furthermore, the objective functions of both approaches are different. 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 last is shown to be NP-Hard, we decided to look for a heuristic solution.

Other problems related to the one presented in this paper are the *Network testbed mapping problem* [13] and the problem defined by Liu *et al.* [9]. In the first approach node mapping is performed considering that each machine can receive a number of virtual nodes determined by testers and no consideration is made about consumption of resources (e.g., 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 virtual machine monitors become widely spread. Besides, both solutions have restrictions in the topology of the real environment they can map, what limits the application of these approaches.

Finally, Singh *et al.* [14] applied a heuristic to map virtual machines and virtual storage systems in data centers. However, their solution, called *VectorDot*, was designed to work in a very restricted environment, not in general environment as was our approach.

3. Problem Description

In this section, we define the problem we are addressing in this paper. The problem consists in finding both a map of virtual machines (guests) to physical machines (hosts) and a map of virtual links between guests to paths among hosts. Description of actual infrastructures that support our approach is presented in Section 3.1, while a formal definition of the problem is presented in Section 3.2.

3.1. Target Environment

The environment we are considering in this work is composed of a cluster of workstations where each node runs a virtual machine monitor (VMM). This cluster may be either homogeneous or heterogeneous regarding its configuration, e.g., the CPU speed and type of each node, amount of RAM memory and so on. The only requirement regarding the configuration of cluster nodes is that they must run the same version of a VMM.

On the top of this real or physical environment, a virtual environment emulating a distributed system is built. The virtual distributed system is composed of a set of virtual nodes (whose parameters are defined by the tester) and a set of virtual network connections among them. Each virtual node in the virtual distributed system corresponds to a virtual machine (guest) running in a cluster node (host).

In some cases, the virtual network may have links between guests whose hosts are not directly connected. For example, if the cluster is linked by a ring network, two non-adjacent hosts are not directly connected, although the virtual machines on them may have a virtual connection. Thus, a virtual link may be composed of more than one physical link.

During the mapping process, it is important to guarantee that hosts have enough resources to support all the virtual machines mapped to them. It is also important to consider that the VMM uses host's resources. Consequently, for each different resource (CPU, memory, storage), the amount of it used by the VMM is deducted from that resource availability prior the mapping.

3.2. Definitions

A cluster of workstations is 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's capacities are defined by functions $proc : C \rightarrow \mathbb{R}$, $mem : C \rightarrow \mathbb{N}$, and $stor : C \rightarrow \mathbb{R}$ that describe the processing capacity, amount of memory, and storage capacity, respectively.

The link's capacity is defined by functions $bw : E_c \rightarrow \mathbb{R}$ and $lat : E_c \rightarrow \mathbb{R}$ 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 virtual machines running in the same host have as much bandwidth as they require to communicate, and the latency of this communication is null.

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

The guests' capacities are defined by functions $vproc : V \rightarrow \mathbb{R}$, $vmem : V \rightarrow \mathbb{N}$, and $vstor : V \rightarrow \mathbb{R}$ that describe the processing capacity, amount of memory, and storage capacity, respectively.

The virtual link's capacities are defined by functions $vbw : E_v \rightarrow \mathbb{R}$ and $vlat : E_v \rightarrow \mathbb{R}$ that describe the bandwidth and the latency respectively.

The mapping problem consists in finding, for each $c_i \in C$ a set $G_i \subseteq V$ where the amount of resources required by all the guests mapped to a host does not exceed the resources of the given host. We are not considering CPU as a constraint of our problem. Instead, CPU usage is used as the variable to be optimized. Moreover, each virtual machine is mapped only once:

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

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

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

For each pair $(vs_j, vd_j) \in E_v$ a sequence $P_j = ((s_1, d_1), (s_2, d_2), ..., (s_p, d_p)), (s_i, d_i) \in E_c$ must be found, in such a way that:

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

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

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

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

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

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

Therefore, our solution grants 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 we consider that 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 virtual machines running on it, delaying the experiment. The objective function we apply tries to balance the utilization of CPU on each host, considering that it can be applied in a heterogeneous environment, where different hosts may have different processing powers. Thus, instead of considering the amount of virtual machines in each host as a load balance metric, we use 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.

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

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

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

In other words, the objective functions tries to assure that there is load balance among hosts, considering resources and network constraints.

4. Proposed Solution

The proposed heuristic solution, dubbed *Hosting-Migration-Networking* heuristic (HMN), consists in the sequential execution of three stages. Each stage is detailed next.

4.1. Hosting

In the first HMN stage, Hosting, a preliminary assignment of guests to hosts is found. In this initial assignment, no consideration is made about load balancing (which is considered in the second stage) or paths (performed in the third stage). Assignment is conduced in such a way that, wherever it is possible, guests with high bandwidth link between them are placed in the same host. It is done in order to reduce the use of physical links, which are one environment constraint. The assignment starts from guests whose links have high-bandwidth. By being mapped first, we increase the chance that these guests are assigned to the same host.

Initially, a list of hosts $c_i \in C$ in descending order of $cpu(c_i)$ and a list of virtual links $(vs_i, vd_i) \in E_v$ in descending order of $vbw((vs_i, vd_i))$ are created. Then, starting from the first element of the links list, hosts are assigned for the guests in the link in the following way:

If both guests vs_i and vd_i were already mapped, the next element in the list is considered. If neither vs_i nor vd_i was mapped, the first host in the hosts list is chosen to host both guests. If they do not fit in the host, the most CPU-intensive guest is assigned to the first host in the list able to receive the guest and the second guest is assigned to the next host which the guest fits in. In both cases resources and CPU availability of the chosen hosts are updated to reflect the assignment, and the host list is sorted again considering the new CPU availabilities.

If one of the guests was already assigned, the unassigned guest is assigned to the same host where the other guest was assigned. If such host does not fit the guest, it is assigned to the first host in the host list able to receive it. After the assignment, resources and CPU availability of the host are updated, and the list is sorted again considering the new CPU availabilities.

If in some moment no host supports an unassigned guest, the heuristic fails. If the Hosting stage succeeds, the heuristic advances to its second stage, namely Migration.

4.2. Migration

The goal of this stage is to increase the load balance in the hosts, through changes in the guests initial assignment. Wherever it is possible, guests initially assigned to high-loaded hosts are reassigned to low-loaded ones. The load-balance factor (Equation 10) of the environment is used to determine the environment degree of imbalance.

At each iteration, the most loaded host is selected as the origin of the migration. The guest chosen to migrate is the one with the smallest sum of bandwidth of links to another guests in the same host, in order to minimize utilization of physical links.

Then, starting from the least-loaded host, the load-balance factor if the migration had just happened is calculated. If this value is smaller than the current load balance factor, and the chosen guest fits in the new host, the reassignment is performed. Otherwise, the next least loaded host is considered. The process is repeated until a reassignment happen or all the hosts are tested.

The whole process is repeated while the load balance factor improves. When no further improvements is possible by migrating a guest from the highest loaded host, the heuristic advances to its last stage—Networking.

4.3. Networking

In this stage, the goal is to find, for each virtual link, a path among real links. The path is built considering the shortest path, constrained by the latency value. The metric used to define the shortest path is the bottleneck bandwidth [16]. The rationale behind the choice of this metric is to keep the links with the largest amount of bandwidth available to map the rest of the links.

Initially, a list of virtual links $(vs_i, vd_i) \in E_v$, sorted in descending order of $vbw((vs_i, vd_i))$ is built. Starting from the first element of the list, a path for the chosen link is built with the modified 1-constrained A*Prune algorithm [8] shown in Algorithm 1.

A*Prune is an algorithm used to QoS routing in networks subject to technical constraints [15]. In our heuristic, A*Prune has been modified to select the path with the greatest bottleneck bandwidth. As distance metric for pruning inadmissible paths, we used the accumulated latency in the Dijkstra path between a given host and the link destination. During the pruning process, links whose available bandwidth are smaller than the required bandwidth are also pruned.

If in some moment a path for a virtual link cannot be found, the heuristic fails. When a path is found for each link in the list, this stage, and the HMN heuristic, finishes.

5. Evaluation

The HMN heuristic was evaluated using simulation. The CloudSim [2] simulation framework was used in the tests. The HMN heuristic was compared with a mapping algorithm that randomly tries to map the guests to hosts and for each link in E_v applies a depth-first search algorithm to find a path connecting the hosts of vs_i and vd_i. The random algorithm fails if it cannot find a valid mapping after 100000 tries.

To allow evaluation of the effect in the mapping of the different stages of HMN heuristic, specially the hosting stage and networking stage, two other heuristics were used in the tests: in the first one, the random algorithm has been

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 $c_i \in C$ do
    ar[$c_i$] $\leftarrow$ length of the Dijkstra path associated to
    latency from $c_i$ to $destination$;
    $set$ $\leftarrow$ ($origin$,$\infty$) (set of feasible paths and their
    bottleneck bandwidths);
    while $set \neq \emptyset$ do
        bestPath $\leftarrow$ path with the greatest bottleneck
        bandwidth, removed from $set$;
        bbw $\leftarrow$ bottleneck bandwidth of bestPath;
        d $\leftarrow$ last element of bestPath;
        if $d$=$destination$ then
            return bestPath;
        end
        for all hosts h connected to d do
            if $h \notin$ bestPath then
                if bw((d,h))$\geq$ $bandwidth$ and
                lat((d,h))+ar[h]$\leq$ $latency$ then
                    $set \leftarrow set \cup (bestPath \cup$
                    $h, min(bw((d,h)), bbw))$;
                end
            end
        end
    end
end
```

used to map guests to hosts and the modified A*Prune has been used to map the link. The other heuristic used in the test applied the hosting algorithm to map guests to hosts and a depth-first search algorithm to map virtual links to paths. Evaluation of these two mixed strategies allowed us to evaluate effectiveness of each stage of the HMN algorithm in the mapping process. For each heuristic, we collected the time to run the experiment, the time to perform the mapping, and the value of the objective function for the given mapping, if a valid mapping was found.

Regarding the workloads, two different use cases were considered: testing of high-level application and testing of low-level applications. The first case encompasses testing 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, a Java virtual machine and so on), which demand large amount of memory and storage. This scenario resembles emulation experiments like the ones presented in our previous work [4].

The second case considers an environment where the objects of tests are, for example P2P protocols. In these tests, there is no need for a virtual machine running several applications demanding many resources. Instead, smaller VMs with only the basic software can be used. Thus, in these tests the virtual machines require less memory and storage. This scenario was based in emulation experiments presented by Quétier *et al.* [11].

5.1. Experiment setup

Table 1 summarizes the experiment setup. The physical environment is composed of two different cluster topologies. In the first one, a 2-D torus topology was used. The second cluster topology was a switched topology, in which hosts were connected to cascade 64-port switches. In both cases, connections between hosts (or connections between a host and a switch) have 1GB of bandwidth and 5ms of latency.

In each test, the cluster topology has been built with the same set of hosts. To represent heterogeneity in the cluster, resources of each of the 40 hosts in the cluster were randomly generated. Host memory varied uniformly between 1GB and 3GB. Storage varied between 1TB and 3TB and CPU capacity between 1000MIPS and 3000MIPS.

The virtual environment configuration was created by a random generator that receives as input the number of guests and network density and generates an output by creating the links between guests and assigning a given amount of resources to each one. Number of resources were generated randomly, based in a normal distribution.

In the high-level experiment workload, the virtual networks were created in such a way that the ratio of guests per host was up to 10:1. Memory of each guest varied uniformly between 128MB and 256MB. Storage of each guest was uniformly distributed between 100GB and 200GB. The MIPS required by each guest varied uniformly between 50 and 100 MIPS. Links between guests had bandwidth defined randomly, with bandwidth values between 0.5Mbps and 1Mbps and latency between 30ms and 60ms.

In the low-level experiment workload, the ratio of guests per host was between 20:1 and 50:1. Memory of each guest varied uniformly between 19MB and 38MB. Storage of each guest was uniformly distributed between 19GB and 38GB. The MIPS required by each guest has been kept varying uniformly between 19 and 38 MIPS. The links between guests had bandwidth defined randomly, with values between 87kbps and 175kbps and latency between 30ms and 60ms.

In both workloads, links between guests were randomly set. The number of links created was determined by the graph density, which was a input parameter to the graph generator. The algorithm used to generate the graph topology guarantees that the output graph is connected.

Table 1. Summary of simulation setup.

	Physical environment	Virtual environment	
		Low-level workload	High-level workload
topology	2-D Torus, Switched	graph, density 0.01	graph, density 0.015-0.025
bandwidth	1Gbps	87kbps–175kbps	0.5Mbps–1Mbps
latency	5ms	30ms-60ms	30ms–60ms
nodes	40	800–2000	100–400
memory	1GB–3GB	19MB–38MB	128MB–256MB
storage	1TB–3TB	19GB–38GB	100GB–200GB
CPU	1000MIPS–3000MIPS	19MIPS–38MIPS	50MIPS–100MIPS

5.2. Results

Each workload has been tested in both clusters. The experiment consisted of execution of each workload with different ratios VM/host repeated 30 times, with the average of the simulation output being used in the result analysis.

Results are presented in Table 2 and Table 3 respectively for objective function and simulation time. In the tables, each column represents a different heuristic: HMN, Random (R), Random with A*Prune (RA) and Hosting with Search (HS). As each scenario has been mapped in two clusters, the results for both clusters are presented in the same line. Table 2 also presents the number of times each heuristic could not find a valid mapping for a specific input. As each scenario in each workload has been simulated 30 times, in the total 480 simulations for each cluster for each heuristic were run.

Each row represents a scenario, and it is described both by the ratio between guests and host (i.e., 10:1 means that the virtual environment contained 10 times more machines than the real environment), and by the virtual environment graph density (i.e., 0.02 means a graph density of 2%). As described previously, for ratios guests/host up to 10:1 we used the high-level experiment workload, while for ratios 20:1 and above we used the low-level experiment workload. These two workloads are separated in the tables by a horizontal line.

As we can notice from Table 2, HMN achieves a reasonable reduction in the objective function, even though its efficacy decreases as the number of guests to be mapped increases. It is an expected behavior, as more guests reduce the chance of migrations during the migration stage.

It is also possible to notice that the main responsible for the success in finding a mapping to the virtual environment is the A*Prune algorithm. It is evidenced both by the large number of failures when hosting is applied without A*Prune and by the success rate of Random mapping with A*Prune. Nevertheless, the great number of failures of HA compared to R is due to the fact that, in the Random approach, both mapping of guests and of virtual links were retried, while in HA only the last one were retried; so, if the initial mapping of guests did not allow a mapping of links, this heuristic fails to find a solution.

Even though a Random heuristic combined with A*Prune is able to find valid mappings, we argue that an initial hosting by network affinity is still the best choice. That is because in the case of mapping virtual environment with a few links with high bandwidth demand, or even with a demand that exceeds the capacity of the real links, HMN is able to produce a valid mapping, while an algorithm that does not group guests with high communication might not be able to find a valid mapping.

Regarding our initial hypothesis that the chosen objective function (Equation 10) reduces the experiment execution time, we found 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.

Figure 1 shows average execution time and standard de-

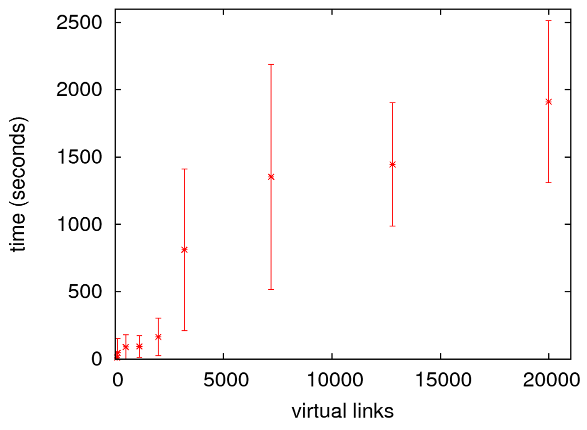

Figure 1. HMN execution time in function of number of virtual links being mapped (torus cluster).

Table 2. Objective function and failures.

	2-D Torus				Switched			
	HMN	R	RA	HS	HMN	R	RA	HS
2.5:1 0.015	573.9	5960.6	5465.2	6838.8	2554.6	6434.5	6209.5	7288.1
5:1 0.015	1241.1	6517.2	5717.7	—	2320.7	6705.3	6442.4	7611.1
7.5:1 0.015	2815.6	—	5734.4	—	3137.8	6744.2	6488	7258.8
10:1 0.015	5619.8	—	5647.35	—	5612.1	6749.8	6395	6845.7
2.5:1 0.02	641.7	5959.8	5600.2	7325.1	2580.6	6492.9	6329.4	7542.1
5:1 0.02	1418.6	6592.7	5727.2	—	2372.7	6440.3	6424.8	7716.8
7.5:1 0.02	2603.1	—	5955.7	—	2929.2	6737.5	6389.7	6966
10:1 0.02	5468.9	—	5590.2	—	5457.8	6715.7	6330.2	6600.6
2.5:1 0.025	567.06	6301.8	5701.2	7612.4	2581.6	6619.2	6431.3	7665
5:1 0.025	1300.3	6398.4	5655.2	—	2351.7	6714.4	6389.3	7480.4
7.5:1 0.025	3039	—	5743.9	—	3298.3	6746.3	6459.3	7347
10:1 0.025	5473.9	—	5636.2	—	5464.9	6981	6373.1	6839.7
20:1 0.01	2003.1	—	8334.2	—	3415.1	8776.7	9108.5	9520.8
30:1 0.01	3878.1	—	8556.5	—	4378.2	8902.9	9268.1	9602.2
40:1 0.01	5614.9	—	8447.7	—	5744	8800.4	9161.3	9265.9
50:1 0.01	8928	—	8483.7	—	8875	9997.2	9244.3	10082.7
Failures	5	322	4	521	5	3	3	5

viation of HMN in function of the number of virtual links being mapped in the torus cluster. Most part of mapping time is spend in the Networking stage to calculate the shortest path of each host c_i to the link destination. This time, however, varied considerably in different simulations of a same scenario. One factor that contributes to this variation is the number of virtual links actually being mapped in the Networking stage, because links whose guests are in the same host are not mapped, as they are handled inside the host. For mapping 2000 guests and 19990 links in a torus cluster, HMN took 30 minutes. Nevertheless, it is an acceptable time, considering that the time to deploy such virtual environment tend to be greater than that [11].

For the switched cluster, the mapping time was less than one second in all scenarios. It happens because in this topology there is only one possible path to each virtual link, namely, the path that goes from the origin host, passes through the switches and reaches the destination host. This small mapping time is an important result, because switched clusters are widely used, and HMN performs well in such widely-available topology.

Because HMN may fail in finding a mapping in scenarios in which the requirements of the virtual system is to close to the resource availability, better heuristics to be applied in these specific cases are subject of current research. While more specific heuristics are not available, HMN heuristic can be successfully applied to solve the mapping problem in general environments.

6. Conclusion and Future Work

Even though there are several distributed systems emulators proposed in literature, no one tackles the problem of automatic mapping of virtual elements to physical ones, considering resources and network constraints, in arbitrary network topologies. In this paper, a modeling of the problem has been presented together with a solution. The proposed heuristic, named Hosting-Migration-Networking (HMN) heuristic, is able to deliver suitable solutions to the problem for a remarkable amount of usage scenarios.

The goal of HMN is finding a mapping that balances the load of the hosts regarding utilization of CPU, respecting the constraints imposed by hosts resources (memory and storage) and also respecting links constraints (bandwidth and latency). Experiments shown that the load balance contributes for reducing the time to run the experiment over the emulated environment.

Nevertheless, there are usage scenarios in which better solutions can be sought. As future works, we intend to develop new heuristics that cover the scenarios where HMN does not perform well. The goal is to offer to the emulator a pool of different heuristics that might be selected according to the emulated scenario. Finally, heuristics for different optimization goals can be developed. For example, one could be interested in a mapping whose goal is to minimize the amount of hosts used in each emulation. Variations in the HMN heuristic in order to attend such different objective functions are also subject of current research.

While more specific heuristics are not available, HMN can be used to solve the mapping problem in general environments. HMN successfully accomplish its goal of providing a heuristic to map virtual machines and virtual links to hosts and physical paths, respectively, in an acceptable time even for large instances of the problem. Considering that availability of large virtualized environments is increasing, automatic methods to determine virtual machines placement, such as HMN, are paramount for the successfully adoption of these platforms as testbeds for testing of

Table 3. Simulation time (seconds).

	2-D Torus				Switched			
	HMN	R	RA	HS	HMN	R	RA	HS
2.5:1 0.015	0.52	0.82	0.81	1.41	0.52	0.8	0.82	1.51
5:1 0.015	0.76	1.33	1.27	—	0.76	1.4	1.32	1.78
7.5:1 0.015	0.96	—	1.85	—	0.96	1.82	1.84	2.22
10:1 0.015	1.52	—	2.37	—	1.52	2.31	2.23	2.35
2.5:1 0.02	0.54	0.83	0.79	1.44	0.54	0.77	0.82	1.48
5:1 0.02	0.77	1.33	1.34	—	0.77	1.3	1.29	1.89
7.5:1 0.02	0.97	—	1.77	—	0.97	1.8	1.73	2.03
10:1 0.02	1.53	—	2.25	—	1.53	2.25	2.25	2.24
2.5:1 0.025	0.55	0.87	0.77	1.48	0.55	0.82	0.83	1.63
5:1 0.025	0.76	1.29	1.3	—	0.76	1.32	1.32	1.85
7.5:1 0.025	0.99	—	1.87	—	0.99	1.76	1.83	2.09
10:1 0.025	1.47	—	2.35	—	1.47	2.36	2.4	2.29
20:1 0.01	0.67	—	1.08	—	0.67	1.04	1.06	1.49
30:1 0.01	0.92	—	1.6	—	0.92	1.56	1.63	2.09
40:1 0.01	1.12	—	2.06	—	1.12	2.06	2.08	2.33
50:1 0.01	1.9	—	2.88	—	1.9	2.95	2.86	3.01

distributed systems through emulation.

Acknowledgments

We would like to thank Marco Netto for their comments on the paper and also for proofreading the text. This work is partially supported by the CAPES PDEE research grant 1185-08-0 and Australian Research Council (ARC) and Department of Innovation, Industry, Science and Research (DIISR).

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] R. Buyya, R. Ranjan, and R. N. Calheiros. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In *7th High Performance Computing and Simulation (HPCS)*, 2009.

[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[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 SBAC-PAD*, 2008.

[5] R. Canonico, P. D. Gennaro, V. Manetti, and G. Ventre. Virtualization techniques in network emulation systems. In *Euro-Par 2007 Workshops: Parallel Processing*, 2007.

[6] 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.

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

[8] G. Liu and K. G. Ramakrishnan. A*Prune: An algorithm for finding K shortest paths subject to multiple constraints. In *20th Annual Joint Conference of the IEEE Computer and Communications Societies*, 2001.

[9] Y. Liu, Y. Li, K. Xiao, and H. Cui. Mapping resources for network emulation with heuristic and genetic algorithms. In *6th International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2005.

[10] A. Oram. *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.

[11] B. Quétier, M. Jan, and F. Cappello. 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.

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

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

[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.

[15] Q. Sun and G. pu Wang. Study on the performance of the A*Prune QoS routing algorithm for intelligent optical networks and its improvements. *The Journal of China Universities of Posts and Telecommunications*, 13(3):65–70, 2006.

[16] A. I. Sundararaj, M. Sanghi, J. R. Lange, and P. A. Dinda. Hardness of approximation and greedy algorithms for the adaptation problem in virtual environments. In *IEEE International Conference on Autonomic Computing*, 2006.

[17] A. I. Sundararaj, M. Sanghi, J. R. Range, and P. A. Dinda. An optimization problem in adaptive virtual environments. *ACM SIGMETRICS Performance Evaluation Review*, 33(2):6–8, 2005.