Quality Estimation of Virtual Machine Placement in Cloud Infrastructures

Jorge López, Natalia Kushik, Djamal Zeghlache

Quality Estimation of Virtual Machine Placement in Cloud Infrastructures

Jorge López, Natalia Kushik, and Djamal Zeghlache

SAMOVAR, CNRS, Télécom SudParis/Université Paris-Saclay, 9 Rue Charles Fourier, 91000 Évry, France
{jorge.lopez, natalia.kushik, djamal.zeghlache}@telecom-sudparis.eu

Abstract. A virtual machine (VM) placement module is a component / part of a cloud (computing) infrastructure, which chooses the *best* host(s) to allocate the requested VMs. In the literature, skewed or biased criteria are often used to determine the correctness of a placement module. Therefore, the quality of existing placement solutions is not always assessed adequately. In this paper, we propose a distance function that estimates the quality of the placement by comparing it with an optimal solution. We show how this distance function is utilized for testing and monitoring the behavior of VM placement implementations. To validate our approach a simulator has been developed and used for estimating the quality of different placement modules running under various scenarios. Preliminary experimental results on VM placement algorithms implemented in widely used platforms, such as OpenStack show that very often VMs are placed very far from the optimal solutions.

Keywords: Quality Estimation, Distance Functions, Integer Linear Programming, Virtual Machine Placement, Testing, Monitoring

1 Introduction

Cloud computing is a computer paradigm, which is based on sharing physical resources. Physical resource sharing enables flexibility, robustness, fast provisioning, fast resource (re-)allocation, etc. Corresponding applications have grown in usage and in demand in recent years; state-of-the-art applications must guarantee fast provisioning (see, for example [17]), and cloud computing aids to achieve such goals. Essentially all *planning* concerning the resource distribution and virtualization is performed by a corresponding *cloud manager*, which needs to be thoroughly tested and verified. One of the principal tasks of a cloud manager is the proper placement of VMs, i.e., choosing *the best* host for a given VM. In the literature, many placement algorithms have been proposed (see, for example [11]); in particular, Masdari et al. presented a comprehensive survey on these algorithms [15].

As VM placement is one of the main tasks of cloud managers, it is critical to properly test the implementations of the corresponding placement algorithms. Currently, researchers mainly focus on evaluating the placement algorithms with

respect to specific criteria rather than testing/monitoring their implementations. Some algorithms are shown to have better performance than others, i.e., they are known to find a corresponding list of hosts for a given set of virtual machines faster. However, such evaluations can be subjective, and moreover optimization criteria can contradict each other. It is arguable if the allocation speed can be considered as a good way to assess the overall correctness of a given placement algorithm. Therefore, the question arises: what is the correct way to assess a virtual machine placement algorithm and corresponding implementation? As mentioned above, in the literature, little attention is paid to this problem. The latter motivates us to propose novel techniques for the placement algorithm assessment as well as for testing its implementation under the assumptions that (i) placement requests are sequentially applied, (ii) the total number of VMs remains unknown, and (iii) limits of the physical resources are finite and known in advance. Assuming a good assessment technique can be found, yet another important question that arises is the following: How to properly verify and monitor the implementation correctness and how to generate *good* test suites for checking the behavior of a given virtual machine placement module?

Therefore, the problem statement is as follows: Given a VM environment, i.e., physical resource limits and VM configurations, and a VM initial placement algorithm to manage the VM placement on this VM environment together with its implementation, one has to (i) assess the correctness / efficiency of the algorithm, (ii) provide methods for the run-time monitoring of the placement implementation, and (iii) derive test suites for the effective assessment of the optimality of the placement implementation. Note that the scope of this work focuses on the initial placement problem, and not in re-allocation / migration or other placement-related problems, such as the selection of the correct overcommit ratio.

To tackle the stated problems, we present a distance function in order to assess the quality of the virtual machine placement algorithm by calculating the distance of the algorithm's *solution* to the optimal one. The introduced distance or metric is further utilized for effective test generation. In fact, we propose to generate the input data for placing so that the VM environment's resource utilization is maximal. To obtain this configuration a proper Integer Linear Programming (ILP) [20] problem is formally stated and solved. Namely, the VM environment information is used to describe an Integer Linear Program that maximizes the VM resource utilization given the cloud infrastructure and VM configuration setup; this can be considered a boundary testing approach [16]. We discuss the use of the distance function to statically verify that a given placement algorithm always returns a result close enough to the optimal one. Likewise, we analyze the use of this function to monitor placement implementations with limited controllability. Finally, in order to show the validity of our approach, we present experimental results that follow a simulation process for different VM environments and placement algorithms.

The paper is structured as follows. Section 2 introduces the required background and the addressed problem. Section 3 contains the related work. Section 4

presents the assessment of a placement module quality by introducing a distance function, while Section 5 is devoted to testing and monitoring placement implementations. Section 6 contains the experimental evaluation, and finally Section 7 concludes the paper.

2 Background

In this section, we briefly describe the necessary concepts used throughout the paper.

2.1 Virtual Machine Placement in cloud infrastructures

In the context of cloud computing, a host machine or simply a *host* is a physical computing device that provides its resources to isolated computing components, i.e., *virtual machines*. A cloud infrastructure is composed of a *set* of heterogeneous interconnected hosts with different resource capacities; commonly such hosts are commodity hardware. When hosts have a shared and common storage, a virtual machine (VM) can be executed in any of such hosts or migrated from one to another. A cloud infrastructure typically contains a cloud manager or orchestrator; the cloud manager is composed of distinct management modules, including, the placement module, which assigns the VMs to the appropriate hosts. A simplified cloud infrastructure is depicted in Fig.1.

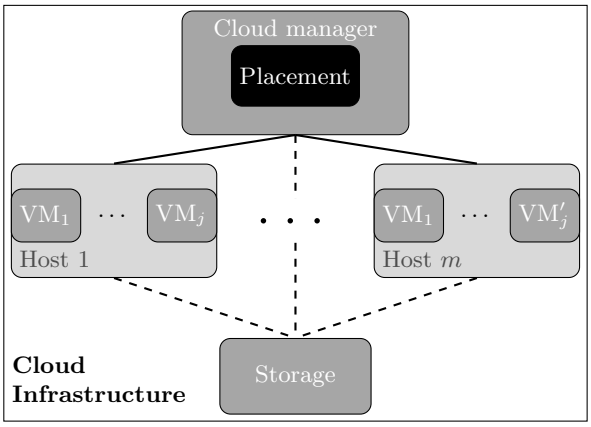

Fig. 1. Cloud computing infrastructure

Hosts and VMs have different limits on physical / virtual resources (or resource parameters). We assume that each VM has the same set of resource parameters as the physical resource parameters of the host executing that VM. For instance, a given host might have 64 CPUs, 96 GB of RAM, and 1 Gbps

of available (network) bandwidth. Note that we consider a coupled architecture, i.e., the VM resources cannot be taken from diverse hosts, such as taking RAM from one host, and CPU from another.

In Fig.2 we depict an example of a cloud infrastructure, which is occupied by different types of virtual machines. This cloud infrastructure is later used as our running example. Consider the arrangement of VMs in the depicted cloud infrastructure, performed by a placement module. If a new request to allocate a VM with 2 CPUs and 2 GB of RAM comes, there is no possibility to complete the request without performing a rearrangement of the VMs in the cloud infrastructure. In fact, VM migrations are considered to be very expensive. Consequently, we consider that a placement algorithm rejects the placement of the VM if migrations are necessary. Further, we consider placement and migrations as different tasks. A placement module is in charge of finding the ***best*** host(s) in the cloud infrastructure to allocate the requested VMs (without migration).

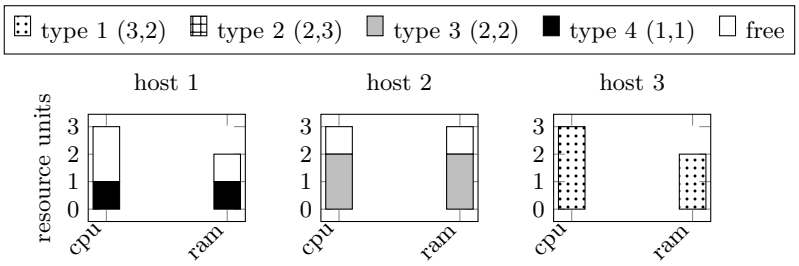

Fig. 2. Cloud computing infrastructure placement / usage

2.2 Integer Linear Programming

In this paper, we provide different techniques for assessing the VM placement algorithm by comparing the solution provided by an Implementation Under Test (IUT) with an optimal one. The latter can be found through a corresponding optimization problem. In this paper, we refer to an Integer Linear Programming (ILP) problem which has the following general form [20]:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0}, \\ & \mathbf{x} \in \mathbb{Z}^n \end{aligned}$$

Where $\mathbf{x}$ is a vector of n non-negative integers, $\mathbf{c}$ and $\mathbf{b}$ are integer vectors, and $\mathbf{A}$ is an integer matrix. The function to maximize is the *cost function* representing the objective (or goal) of the optimization problem. The remainder of

the problem formulation presents the *constraints* of the problem, i.e., what the maximization of the cost function is subject to. Finally, the ILP problem represents in most cases a typical combinatorial optimization problem; the latter means that there exists a finite number of possible solutions, and the solutions must be integer-valued. A given instance of the optimization problem is the tuple (F, c), where F is a domain of feasible points, and c is a cost function such that $c : F \mapsto \mathbb{Z}_+$. The solution of the optimization problem is to find $f \in F$ such that $c(f) \geq c(y); \forall y \in F$; in this case, the point f is a globally optimal solution.

In the literature, algorithms to solve linear programming problems have been studied since the 1940s. One of most popular ones is the Dantzig's Simplex algorithm [7]. Correspondingly, to solve ILP problems, the cutting plane methods based on Gomory's algorithm [10] can be applied. Modern tools for solving instances of ILP problems as Gurobi [19], use such cutting plane methods and others. It is important to mention that even if the ILP problem is NP-hard, in practice, problems with hundreds or thousands of variables can be easily solved in a few seconds.

3 Related Work

In this section, we briefly discuss the existing criteria and methodologies to determine the quality of a virtual machine placement module, i.e., IUT. To some extent, all VM placement schemes found in the literature try to assess the quality of the proposed algorithm with respect to a number of criteria, commonly favoring their implementation. A large number of research papers proposing novel algorithms tend to focus on the execution time. The approaches typically model the VM placement problem as the known bin-packing problem (or similar) [1, 3, 8], which is NP-hard. The objective of bin packing is to minimize the number of containers of size V necessary to *pack* a fixed number of objects of different sizes. However, in a cloud infrastructure, VM (allocation) requests come at different time instances and the requested VM types are not known in advance. Therefore, solving the bin packing problem for a batch of requested machines guarantees optimality in resource utilization for the allocated batch, not guaranteeing the overall optimality in the resource utilization. For that reason, many researchers have devoted their efforts to propose approximation algorithms [22] and they estimate the quality of their algorithms with respect to their time complexity. The latter overlooks any other quality measures, including the optimality of the resource utilization.

Nonetheless, works devoted to categorizing the algorithms with respect to their optimization goals have also been published. Such works are mostly surveys, an interested reader can refer to some comprehensive works in [21] and [15]. The categories used to classify the papers, i.e., the optimization goals provide a good notion of the important aspects to evaluate a VM placement IUT. Resource utilization and energy consumption are among the most common criteria used to evaluate a placement implementation, aside from the time / performance discussed above. Nonetheless, due to the nature of such works the quality es-

timation is not the target of their research. Therefore, little attention is paid to the the quality estimation and comparison of different placement algorithms. As a consequence, such works do not consider how to effectively test any VM placement implementation.

In [9], the authors describe in detail what are the relevant criteria to do a comparative analysis of VM placement algorithms, their classification, and conclude about their advantages. The discussion (Section 8 of their work) entitled "Comparison of VMP techniques" is interesting as the authors make conclusions based on the characteristics and properties of the proposed solutions. However, even if the conclusions are convincing, there is no formal proof nor experimental evaluation to support the statements. Furthermore, the discussion focuses only on a small number of VM placement schemes.

Finally, there exist some works, which aim to evaluate the quality of the placement algorithms using a set of metrics over the resulting configurations of a given set of inputs, similar to our proposal. In [5] and [14], the authors employ CloudSim [2, 4], a toolkit for the simulation of cloud computing environments. Both of the previously mentioned works use a similar set of metrics: (i) SLA violation, the average percentage of time which a host CPU utilization was over 100%; (ii) Performance degradation due to VM migrations, and finally (iii) Energy consumption (proposed only in [14]). Another common characteristic of both works is the fact that they employ existing data sets and do not focus on test generation.

To the best of our knowledge, there exists no work in the literature that presents different methods for the quality estimation of a VM placement module based on measuring the distance from an optimal solution. Furthermore, we are not aware of the methods for effective test suite generation which can be used in conjunction to the metrics / distance functions; similarly, no methods to derive properties for monitoring and verification of the IUT at run-time are known to the authors. Finally, when evaluating the quality of placement IUT, the approaches assume a fixed number (usually 2 or 3) of virtual machine parameters, e.g., they consider only CPU and RAM, differently from our approach, which generalizes the notion to multidimensional vectors.

4 Virtual Machine Placement Quality Estimation

In this section, we present an approach to evaluate the quality of a virtual machine placement module in a given cloud infrastructure. The approach is based on *distance functions*. The introduced distances or metrics can be further used for checking functional properties of the VM placement modules.

4.1 Definitions and Notations

Let h be a host in a cloud infrastructure. We represent h as a tuple referring to its physical resource limits, i.e., $h \in \mathbb{Z}_+^p$, where p is the total number of physical resource parameters. We assume each physical resource parameter of a host

is expressed as a multiple of an elementary resource unit (CPU, RAM, storage, etc., are multiples of their respective elementary resource unit), i.e., the elements are normalized by their respective reference unit. As an example, a host with 3 CPUs and 1GB of RAM whose minimal unit values are *1 CPU*[1] and *512MB of RAM*, can be represented as the pair $(3, 2)$. In practice, virtual infrastructure managers overuse the available physical resources under the assumption that not all virtual machines use all available resources at all time [18]. We assume that the overcommit factor is taken into consideration for all values of all hosts' physical resource parameters. Consider the previous host $(3, 2)$, virtual infrastructure managers might consider overusing the RAM of this host by a factor of two, in this case, the host's physical limits are $(3, 4)$.

As previously stated, a VM is assumed to have the same virtual resource parameters as the physical resource parameters of the cloud infrastructure. Therefore, we define a VM in the same manner as a host. Let vm be a virtual machine represented as a tuple of its virtual resource limits, i.e., $vm \in \mathbb{Z}_+^p$, where p is the total number of physical resource parameters.

A cloud infrastructure CI being a collection of hosts is represented as a tuple $CI = \langle h_1, h_2, \ldots, h_m \rangle$, where $h_i = (h_{i_1}, h_{i_2}, \ldots, h_{i_p}) \in \mathbb{Z}_+^p$ and m represents the total number of hosts in the cloud infrastructure. As an example, consider the cloud infrastructure with CPU and RAM parameters depicted in Fig.2. The cloud infrastructure can be represented as the following object: $CI = \langle (3, 2), (3, 3), (3, 2) \rangle$.

Each cloud infrastructure has an associated and finite number of possible VM configurations, i.e., a fixed amount of possible VMs with *different* virtual resource requirements. We denote the VM configuration setup (VC) of n different configuration types, similarly to a cloud infrastructure: let VC be a virtual machine configuration setup referred to as a tuple of n elements, where each element is a VM, i.e., $VC = \langle vm_1, vm_2, \ldots, vm_n \rangle$, where $vm_j = (vm_{j_1}, vm_{j_2}, \ldots, vm_{j_p}) \in \mathbb{Z}_+^p$, and $\forall j, j', vm_j \neq vm_{j'}$ if $j \neq j'$, n is the number of distinct possible VM configurations (often also seen as VM types). As an example, consider the VM configuration setup depicted in Fig.2, this configuration is represented by the following object: $VC = \langle (3, 2), (2, 3), (2, 2), (1, 1) \rangle$.

Given a virtual machine configuration, and a cloud infrastructure, a *placement configuration* represents which type of virtual machines are allocated at each given host. Formally, we denote a Placement Configuration PC in the following manner: PC can be represented by the matrix $PC_{m \times n}$, $pc_{ij} \in \mathbb{N} \cup \{0\}$, where pc_{ij} represents the number of VMs with a vm_j configuration placed on the h_i host.

As an example, consider the cloud infrastructure and virtual machine configuration setup depicted in Fig.2, the matrix $PC_{m \times n} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$ can be interpreted as: in the first host $h_1 = (3, 2)$ one VM of the fourth configuration type, i.e., $vm_4 = (1, 1)$ is allocated; in the second host $h_2 = (3, 3)$ one VM of the third

[1] Number of cores in compute-centric applications

configuration type, i.e. $vm_3 = (2, 2)$ is placed (or allocated), and finally, on the third host $h_3 = (3, 2)$ a VM of the first configuration type $vm_1 = (3, 2)$ is placed.

A virtual machine placement request is a *batch* or *list* of VMs of different configuration types. Formally a request r can be denoted as follows. Let VC be a virtual machine configuration setup. A virtual machine placement request r is a tuple of n elements, in which each element represents the requested number of VMs of the corresponding type in VC. Particularly, $r = \langle q_1, q_2, \ldots, q_n \rangle$, where $q_j \in \mathbb{N} \cup \{0\}$; r represents a request for a collection of VMs, where q_j is the number of VMs of the type vm_j, for $j = 1, 2, \ldots, n$. In a straightforward manner, a request can be extended to a request sequence $\alpha = r_1 r_2 \ldots r_l$. Given a request sequence α, α_{i_j} denotes the requested quantity of VMs of the type j in the i-th request. As an example, a virtual machine placement request $\langle 0, 0, 0, 2 \rangle$ for the VM configuration setup $VC = \langle (3, 2), (2, 3), (2, 2), (1, 1) \rangle$ as depicted in Fig.2, is a request for two VMs of the fourth type, i.e. $vm_4 = (1, 1)$.

A placement algorithm $\mathcal{A}$ takes as inputs a cloud infrastructure CI, a virtual machine configuration setup VC, an initial placement configuration PC, a request r, and produces as an output a new placement configuration PC'. The behavior of $\mathcal{A}$ can be extended to request sequences by considering the output of $\mathcal{A}$ as the initial configuration PC for the next request in the sequence α.

4.2 Placement Quality Evaluation

As mentioned in Section 3, there exist some works [5, 9, 14], which propose different criteria to evaluate the quality of placement implementations. To some extent, we question some of the criteria, for instance, SLA violation reflects not an incorrect placement, but an incorrect definition of the limits of the hosts. Then, what is a good criterion to determine the quality of a placement algorithm? From the functional point of view *an efficient resource-aware placement scheme tries to optimally place VMs on the PMs (Physical Machines i.e., hosts) such that the overall resource utilization is maximized* [15]. From the previous statement three important conclusions can be made: (i) the optimal placement maximizes the resource utilization, (ii) the quality of the placement implementation decreases as it moves away from the *optimal placement*, and (iii) the correctness of an algorithm can be measured with respect to optimality.

To define a proper distance function, we first introduce the concept of overall resource utilization. For the lack of a proper definition, we define it as follows: the overall resource utilization measures the total number of resource units taken by the allocated virtual machines in the cloud infrastructure.

Definition 1. *Given a cloud infrastructure, a virtual machine configuration and a placement configuration, the overall resource utilization f is a function defined as: $f(CI, VC, PC) = \sum_{i=1}^{m} \sum_{j=1}^{n} (pc_{ij} * \sum_{k=1}^{p} vm_{j_k})$. Note that the image of f is the set of natural numbers (including zero) $\mathbb{N} \cup \{0\}$.*

An intuitive explanation of the construction of f is the following. The overall resource utilization is the sum of all resources utilized in all hosts of the infrastructure CI, i.e. $f(CI, VC, PC) = \sum_{i=1}^{m} f'(h_i, VC, PC)$, where $f'(h_i, VC, PC)$

is the resource utilization on the host i. The resource utilization on the host i is the sum of all resources taken by all VMs of all different types allocated in the host, i.e., $f'(h_i, VC, PC) = \sum_{j=1}^{n} f''(h_i, vm_j, PC)$. The sum of all resources taken by a VM is $\sum_{k=1}^{p} vm_{j_k}$ and correspondingly the quantity of VMs of a type vm_j allocated in a host h_i is pc_{ij}. Therefore, $f''(h_i, vm_j, PC) = pc_{ij} * \sum_{k=1}^{p} vm_{j_k}$.

An important remark is that a placement configuration can be obtained from the simulation of an implementation of $\mathcal{A}$ and a request sequence α. Given α, we denote the placement configuration obtained from the simulation of $\mathcal{A}$ as $PC = sim(\mathcal{A}, \alpha)$.

As mentioned above, the distance for the algorithm assessment depends on an optimal placement. We define an optimal solution with respect to the maximal resource utilization in the following manner:

Definition 2. *The algorithm $\mathcal{O}$ is optimal with respect to the overall resource allocation, if it holds that $f(CI, VC, sim(\mathcal{O}, \alpha)) \geq f(CI, VC, sim(\mathcal{A}, \alpha))$, $\forall \mathcal{A} \in \mathcal{P}$, where $\mathcal{P}$ is the set of all possible placement algorithms.*

To measure the distance between the overall resource utilization of a given algorithm or its implementation and an optimal resource utilization, we define the distance as a simple Euclidean distance[2], namely:

Definition 3. *A resource utilization distance d is defined as the metric $d : \mathcal{N} \cup \{0\} \times \mathcal{N} \cup \{0\} \mapsto \mathcal{N} \cup \{0\}, d(x, y) = |x - y|$, where x is the resource utilization of a given implementation and y is the resource utilization of an optimal algorithm $\mathcal{O}$.*

Intuitively, this distance expresses how far the overall resource utilization is from an optimal solution. In the following sections, we highlight the usefulness of this definition.

5 Testing, Monitoring, and Validating Placement Modules

In this section, we discuss how the proposed VM placement quality evaluation approach can be used when testing placement modules in cloud infrastructures. In this case, the corresponding placement module represents the implementation under test (IUT) and the conclusion about its quality is made based on the value of the distance d from an optimal solution.

5.1 Boundary Test Case Generation for Placement Modules

Given an IUT (a placement module), we propose deriving a series of requests that produce the maximum possible resource utilization for the given cloud infrastructure (and VM configuration). To achieve this, an appropriate ILP problem

[2] The obtained overall resource utilization is a natural number, therefore Euclidean or other metrics can be considered

can be formulated and solved to find the corresponding values bringing the value of the function f to its maximum. In this case, the ILP formulation is somewhat straightforward. The cost function to optimize is the resource utilization function f, and the goal is the maximization of that function. The constraints are the limits of the hosts in the cloud infrastructure. The unknowns are the number of VMs of different types on each host, representing the placement configuration matrix. The general form of the problem is the following:

$$\begin{aligned} \text{maximize} \quad & f = \sum_{i=1}^{m} \sum_{j=1}^{n} \left(pc_{ij} * \sum_{k=1}^{p} vm_{j_k} \right) \\ \text{subject to} \quad & \sum_{j=1}^{n} vm_{j_k} * pc_{ij} \leq h_{i_k}; \forall i = 1, \ldots, m; k = 1, \ldots, p, \\ & \mathbf{PC} \geq \mathbf{0}, \\ & pc_{ij} \in \mathbb{Z}, \forall i = 1, \ldots, m; j = 1, \ldots, n \end{aligned} \tag{1}$$

For the cloud infrastructure and VM configuration setup of the running example, the solution of the following ILP problem provides maximal resource utilization:

$$\begin{aligned} \text{maximize} \quad & f = 5pc_{11} + 5pc_{12} + 4pc_{13} + 2pc_{14} + 5pc_{21} + 5pc_{22} + 4pc_{23} + 2pc_{24} + \\ & 5pc_{31} + 5pc_{32} + 4pc_{33} + 2pc_{34} \\ \text{subject to} \quad & \\ & 3pc_{11} + 2pc_{12} + 2pc_{13} + 1pc_{14} \leq 3 \text{ (Max CPU host 1)}, \\ & 2pc_{11} + 3pc_{12} + 2pc_{13} + 1pc_{14} \leq 2 \text{ (Max RAM host 1)}, \\ & 3pc_{21} + 2pc_{22} + 2pc_{23} + 1pc_{24} \leq 3 \text{ (Max CPU host 2)}, \\ & 2pc_{21} + 3pc_{22} + 2pc_{23} + 1pc_{24} \leq 3 \text{ (Max RAM host 2)}, \\ & 3pc_{31} + 2pc_{32} + 2pc_{33} + 1pc_{34} \leq 3 \text{ (Max CPU host 3)}, \\ & 2pc_{31} + 3pc_{32} + 2pc_{33} + 1pc_{34} \leq 2 \text{ (Max RAM host 3)}, \\ & pc_{11}, pc_{12}, pc_{13}, pc_{14}, pc_{21}, pc_{22}, pc_{23}, pc_{24}, pc_{31}, pc_{32}, pc_{33}, pc_{34} \geq 0, \\ & pc_{11}, pc_{12}, pc_{13}, pc_{14}, pc_{21}, pc_{22}, pc_{23}, pc_{24}, pc_{31}, pc_{32}, pc_{33}, pc_{34} \in \mathbb{Z} \end{aligned}$$

A solution to the ILP problem (arranged as a matrix) is: $\left(\begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 \\ 1 & 0 & 0 & 0 \end{smallmatrix}\right)$.

As mentioned above, we are interested in maximizing the resource utilization for the given CI and VC. For this reason, we provide a test sequence (or a test suite) that leads to the maximal resource utilization of a particular cloud infrastructure, i.e., when no more VMs can fit. We furthermore assume that such test generation technique can be treated as a boundary [16] one, by assigning the boundary values on the maximal resource utilization.

Once *the fullest* placement configuration is obtained, a test suite needs to be derived from this placement configuration. A test sequence, in this case, is a request sequence α. To determine the order and the grouping of VMs on each request, we introduce the in-order conjecture.

In-order conjecture: Given a sequential list of individual elements (or VMs) with different resource utilization to allocate into the containers (or hosts), sequentially allocating the elements in the list sorted in ascending order by their overall resource utilization is the most difficult arrangement to allocate.

The reasoning behind the in-order conjecture is that as the containers fill up available resources, independently from the allocation strategy. As a consequence, allocating the largest elements at the end is only possible if the allocation chooses the *best* configuration. As a support to this claim, it can be seen that heuristic methods for greedy allocation algorithms obtain their best results when allocating in reverse order [6, 12].

Due to the in-order conjecture, to thoroughly test the placement modules under stressful conditions, each request contains a single VM and the requests are arranged in ascending order with respect to their overall resource utilization, namely $u(vm_j) = \sum_{k=1}^{p} vm_{j_k}$.

In the running example, $\alpha = \langle 0,0,0,1\rangle\langle 0,0,0,1\rangle\langle 0,0,0,1\rangle\langle 1,0,0,0\rangle\langle 1,0,0,0\rangle$ is a test suite of one sequence. This is a request for three VMs of type 4, and two VMs of type 1. This test suite can be further applied to an IUT of an algorithm $\mathcal{A}$, and the verdict about its quality can be made based on the distance function d. As the optimal algorithm / implementation $\mathcal{O}$ we consider an idealized algorithm (*oracle*) that provides the solution PC to the ILP problem.

The set of constraints for the ILP problem can differ. In particular, two cases are possible: (i) all the hosts are empty when the test sequence represented by a set of VM placement requests, is applied; (ii) the IUT has an initial placement configuration which corresponds to already executed sequences of requests. In case (i), we in fact assume that the VM placement implementation is initialized, i.e., before any test sequence we are allowed to apply a corresponding reliable RESET. If that is the case, then the test sequence that brings the IUT to *full* hosts resource utilization should be applied. In case (ii), the VM placement module works with an initial placement configuration PCI. In other words, the hosts or containers are currently executing (hosting) VMs. This fact can be interpreted by an absence of the corresponding RESET input. In other words, the IUT is not *switched off* nor tested in a complete isolation. In fact, it receives the boundary test suite given its current configuration (PCI). The latter means that the user requests that have been previously implemented, are not lost. In order to derive the proper test sequence one can adapt the set of constraints listed above. We still maximize the function f, however we now assume that some of the unknowns of the ILP are bounded by a positive integer. This fact can be expressed by the following system of linear constraints:

$$pc_{ij} \geq pci_{ij}; \forall i = 1, \ldots, m; j = 1, \ldots, n$$

In order to obtain a test suite, the placement configuration to be considered is the matrix $PC - PCI$, where PC is the matrix obtained from the solution of the ILP problem. Consider the placement in the running example, assume that the initial configuration is exactly as shown. $PC - PCI = \left(\begin{smallmatrix} 0&0&0&2\\ 0&0&1&1\\ 1&0&0&0 \end{smallmatrix}\right) - \left(\begin{smallmatrix} 0&0&0&1\\ 0&0&1&0\\ 1&0&0&0 \end{smallmatrix}\right) = \left(\begin{smallmatrix} 0&0&0&1\\ 0&0&0&1\\ 0&0&0&0 \end{smallmatrix}\right)$. The obtained test suite is the following: $\alpha_I = \langle 0,0,0,1\rangle\langle 0,0,0,1\rangle$.

5.2 Static code/algorithm analysis

Given the resource utilization function f together with the distance d measuring the optimality of a given solution, one can use various static code/algorithm analysis techniques in order to estimate their quality. Such analysis can, on one hand include a random simulation of the placement algorithm $\mathcal{A}$, and on the other hand can allow to perform the backtracking to discover if a given value v of a distance can eventually be reached. In particular, given the algorithm $\mathcal{A}$ that is implemented in the VM placement module under test, the value v of the distance d between the computed resource utilization for the output of algorithm $\mathcal{A}$ and the optimal algorithm $\mathcal{O}$ that corresponds to the proper ILP solution, the question arises: does there exist an input α to the algorithms $\mathcal{A}$ and $\mathcal{O}$, such that $d(f(CI, VC, sim(\mathcal{A}, \alpha)), f(CI, VC, sim(\mathcal{O}, \alpha)) \geq v$. This problem can be represented as a formal verification or model checking issue and the possibility and complexity of solving such issue essentially depend on the definition of the functions d and f. In our case, the way that the distance from an optimal solution as well as the resource utilization function are defined, allows to reduce the problem to a simpler one, using the following system of linear inequalities over the natural numbers:

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\left((pc_{ij} - pco_{ij}) * \sum_{k=1}^{p} vm_{j_k}\right) \geq v \text{ Or } \sum_{i=1}^{m}\sum_{j=1}^{n}\left((pc_{ij} - pco_{ij}) * \sum_{k=1}^{p} vm_{j_k}\right) \leq -v$$

Here, $sim(\mathcal{O}, \alpha) = PCO$ and $sim(\mathcal{A}, \alpha) = PC$.

If this problem has a solution then there exists a set of requests taking the cloud infrastructure from the initial configuration to that one where the distance between the current solution and an optimal one is greater or equal to v.

5.3 Dynamic VM placement execution

We assume that dynamic VM placement quality estimation involves the execution of the system under test, i.e., the placement module itself. In this case, the distance function d can serve as an *oracle* that allows either to take the decision about the optimality of the system under test, or can help to provide an appropriate alarm during the system monitoring.

The distance function d can also be used when the IUT, i.e., the VM placement module, has a limited controllability. Consider the case when no inputs can be applied to the IUT and the tester can only observe the user VM placement requests as well as the resulting configurations. Whenever an input sequence $\alpha = r_1 \dots r_l$ is observed, one can compute the value of the f function for the current resource utilization, after the last request r_l was processed. In this case, for the given input $\alpha = r_1 \dots r_l$ the optimal solution of $\mathcal{O}$ can be calculated. Together with that, the distance between the resource utilization of the provided solution $f(VI, VC, sim(\mathcal{A}, \alpha))$ that is observed by the tester and the optimal solution $f(VI, VC, sim(\mathcal{O}, \alpha))$ can be computed via the application of the function d. Similar to the first case (Section 5.2), given the constant v representing

the largest allowed distance, whenever the computed distance is greater or equal to v an alarm signaling this fact can be produced.

We note that for optimization and scalability reasons some additional calculations can be performed in advance, before the monitoring itself. For example, one can collect specific critical input sequences for which the resulting value of the d function must be computed at run-time. For the set of such user requests, a tester can pre-calculate the optimal solutions and the corresponding values of the function f. Whenever such request sequence is observed, the tester only compares the value v with the given distance and signals when the VM placement produces the solution farther than required. If the value v remains constant, one can use a combination between the approaches from 5.2 and 5.3. If the verification analysis can return all possible user requests that lead to the result which is *very far* from the optimal solution, w.r.t. the defined f and d functions, these sequences of requests can be stored additionally. Whenever a preamble of any of such sequence is observed during the monitoring, a proper warning can be produced regarding the distance to the optimal resource utilization. Consider the running example for $v = 4$, where the request sequence $\alpha = \langle 0,0,0,1\rangle\langle 0,0,1,0\rangle\langle 1,0,0,0\rangle$ has been observed. If a new request $\langle 1,0,0,0\rangle$ is observed, $f(VI, VC, sim(\mathcal{A}, \alpha)) = 11$. However, $f(VI, VC, sim(\mathcal{O}, \alpha)) = 16$ and thus, a monitoring alert is produced.

Finally, a heuristic to compute the optimal resource utilization can be used in order to provide verdicts at run-time. If adding constraints similar to the non-initialized resource utilization, but over the sum of all different requested types of VMs have a feasible solution, the maximal resource utilization equals the sum of all requested resources. Thus, the usage of this heuristic can essentially improve the monitoring by reducing the complexity of the corresponding ILP problem.

6 Experimental Evaluation

In order to validate our approach experimental evaluation is presented. We propose a methodology, which is based on three stages, namely: (i) The generation of test suites for different cloud infrastructures and VM configuration setups using the boundary testing approach; (ii) The simulation of the obtained test suite as a request sequence for different placement algorithm implementations (IUTs); and finally (iii) Evaluating the quality of the IUTs using the defined distance function d.

Experimental Setup. To generate the test suites, the Gurobi [19] tool was employed for solving the ILP problems. After obtaining the proper values for the placement configuration semi-manual processes were involved, namely translating the solutions into the test suites and executing the simulator against them.

In this work, the simulator has been developed in order to perform the experimental evaluation. It is an ad-hoc simulator written in Java. More information about the tool, including its source code can be found in [13]. Currently, the simulator implements two algorithms. The first algorithm is a greedy *first fit* (FF)

algorithm. The FF algorithm places the requested VM into the first host that is able to fit it. The second algorithm is an *available random* (AR) algorithm. The AR algorithm pre-filters the hosts in the cloud infrastructure. The resulting list of hosts from the AR algorithm contains the hosts that can fit the requested VM. Later on, the selection of the host is done with a uniformly distributed random choice. An interesting note on the AR algorithm is that AR with additional filters is the algorithm used in the placement module of the well-known OpenStack virtualization platform [18]. Due to the fact that random algorithms might generate different placement configurations, thus different distances can be obtained, our simulator runs each simulation 100 times and computes the average distance. Since we focus on the initial placement problem and not on re-allocation / migration, when an unfulfillable request comes the algorithms reject the request, and continue accepting the following requests (if any).

Experimental Results. To present our obtained results, we first summarize the test cases generated using the Gurobi software (Table 1). As it can be seen, the test suite generation is quite fast. All the test cases have been introduced into the simulator. All simulations and test generation were run on a MacBook Pro with a 2.3 GHz Intel Core i5 CPU, and 16 GB of RAM @ 1333 MHz DDR3.

Table 1. Test Cases

TC	Environment	ILP details	Sol. time	Comments
1	m=2, n=4, p=2	8 unknowns, 4 constraints	0m0.018s	A very small example.
2	m=3, n=4, p=2	12 unknowns, 6 constraints	0m0.017s	The paper's running example. Gurobi input and solution files available in [13].
3	m=3, n=4, p=2	12 unknowns, 6 constraints	0m0.020s	Increased hosts' capacity, considering real server capacity.
4	m=9, n=4, p=2	36 unknowns, 18 constraints	0m0.048s	Real hosts' capacity, bigger cloud infrastructure.

In Table 2, we summarize the simulation results. It can be clearly seen that the distances (from an optimal) of the FF algorithm are smaller than the distances of the AR algorithm for all test cases. Further, the simulation is also performed faster. These results are in fact expected since the FF algorithm does not perform any pre-selection of hosts. As a conclusion, the quality of the algorithm used in the widespread OpenStack placement module is lower than a very simple first fit algorithm.

7 Conclusions

In this paper, we have proposed a distance function, which allows to effectively assess the quality of a placement module implementation. The metric being introduced measures the distance between the IUT resource utilization and an

Table 2. Simulation Results

TC	$\|\alpha\|$	FF Avg. d	AR Avg. d	FF Sim. time	AR Sim. time
1	4	0	4.45	0m0.005s	0m0.025s
2	5	5	5.5	0m0.007s	0m0.018s
3	78	35	79.25	0m0.07s	0m0.087s
4	270	210	230.95	0m0.219s	0m0.443s

optimal resource utilization. Furthermore, different approaches for testing, monitoring, and verification of the IUT, represented by a VM placement module in a cloud infrastructure, have been proposed; these approaches are coherent with the proposed distance. In order to validate our approach, a simulator of such infrastructures has been developed. Interesting results have been obtained, including the assessment of the placement algorithm used in the widespread OpenStack platform, which is very far from the optimal with respect to resource utilization.

This paper presents an initial approach to effectively assess the quality of placement modules. Different aspects of our approach can be improved and extended. First, we plan to generate test suites for real case studies to further validate our approach. Also, our simulator can be expanded to (i) parse the cloud infrastructure and VM configuration setup from a defined file format, and (ii) based on the configurations, automatically generate the test suites and simulate the results by integrating the ILP solution into our tool. Furthermore, we intend to study other criteria for the placement optimality, for example, energy consumption, performance, etc., and propose and calculate the distances for them. One of the interesting questions is in fact the study of multi-criteria evaluation of the optimality using the listed parameters. The previously mentioned extensions and enhancements represent the future work for the short term.

Acknowledgments. The authors would like to thank Professor Nina Yevtushenko for fruitful discussions. The results obtained in this work were partially funded by the Celtic-Plus European project SENDATE, ID C2015/3-1; French National project CARP (FUI 19); Bilateral contracts with Orange Labs.

References

[1] Babu, K.R.R., Samuel, P.: Virtual machine placement for improved quality in iaas cloud. In: 2014 Fourth International Conference on Advances in Computing and Communications. pp. 190–194 (Aug 2014)

[2] Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurrency and Computation: Practice and Experience 24(13), 1397–1420 (2012), http://dx.doi.org/10.1002/cpe.1867

[3] Bonde, D.: Techniques for Virtual Machine Placement in Clouds. Master's thesis, Indian Institute of Computer Science and Engineering (2010)

[4] Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience 41(1), 23–50 (2011), http://dx.doi.org/10.1002/spe.995
[5] Chowdhury, M.R., Mahmud, M.R., Rahman, R.M.: Implementation and performance analysis of various vm placement strategies in cloudsim. Journal of Cloud Computing 4(1), 20 (2015), http://dx.doi.org/10.1186/s13677-015-0045-5
[6] Culberson, J.C., Luo, F.: Exploring the k-colorable landscape with iterated greedy. Cliques, coloring, and satisfiability: second DIMACS implementation challenge pp. 245 – 284 (1996)
[7] Dantzig, G.: Linear programming and extensions. Princeton university press (1963)
[8] Fei, M., Feng, L., Zhen, L.: Multi-objective optimization for initial virtual machine placement in cloud data center. Journal of Information &Computational Science 9(16), 5029–5038 (2012)
[9] Gohil, B., Shah, S., Golechha, Y., Patel, D.: A comparative analysis of virtual machine placement techniques in the cloud environment. International Journal of Computer Applications 156(14), 12–18 (Dec 2016)
[10] Gomory, R.E.: Outline of an algorithm for integer solutions to linear programs. Bulletin Amer. Math. Soc 64(5) (1958)
[11] Khebbache, S., Hadji, M., Zeghlache, D.: Scalable and cost-efficient algorithms for vnf chaining and placement problem. In: 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN). pp. 92–99 (March 2017)
[12] Lewis, R.: A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. Computers & Operations Research 36(7), 2295 – 2310 (2009)
[13] López, J.: Vmplacementsim. Web Resource, https://github.com/jorgelopezcoronado/VMPlacementSim
[14] Mann, Z.A., Szab, M.: Which is the best algorithm for virtual machine placement optimization? Concurrency and Computation: Practice and Experience 29(10), e4083–n/a (2017), e4083 cpe.4083
[15] Masdari, M., Nabavi, S.S., Ahmadi, V.: An overview of virtual machine placement schemes in cloud computing. Journal of Network and Computer Applications 66, 106–127 (2016)
[16] Mathur, A.P.: Foundations of Software Testing. Addison-Wesley Professional, 1st edn. (2008)
[17] Mechtri, M., Benyahia, I.G., Zeghlache, D.: Agile service manager for 5g. In: NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium. pp. 1285–1290 (April 2016)
[18] OpenStack: Deep dive: Virtual machine placement in openstack:. Web Resource, https://platform9.com/blog/virtual-machine-placement-openstack/
[19] Optimization, G., et al.: Gurobi optimizer reference manual. URL: http://www. gurobi. com 2, 1–3 (2012)
[20] Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1982)
[21] Pires, F.L., Barán, B.: Virtual machine placement literature review. CoRR abs/1506.01509 (2015), http://arxiv.org/abs/1506.01509
[22] Vazirani, V.V.: Approximation Algorithms. Springer-Verlag New York, Inc., New York, NY, USA (2001)