Gomez-Rodriguez, M.A., Sosa-Sosa, V.J., Carretero, J. et al. CloudBench: an integrated evaluation of VM placement algorithms in clouds. *J Supercomput* **76**, 7047–7080 (2020).

DOI: 10.1007/s11227-019-03141-9

# CloudBench: an integrated evaluation of VM placement algorithms in clouds

**Mario A. Gomez-Rodriguez· Víctor J. Sosa-Sosa · Jesus Carretero · José Luis González**

**Abstract**
A complex and important task in the cloud resource management is the efficient allocation of virtual machines (VMs), or containers, in physical machines (PMs). The evaluation of VM placement techniques in real-world clouds can be tedious, complex, and time-consuming. This situation has motivated an increasing use of cloud simulators that facilitate this type of evaluations. However, most of the reported VM placement techniques based on simulations have been evaluated considering one specific cloud resource (e.g., CPU), whereas values often unrealistic are assumed for other resources (e.g., RAM, awaiting times, application workloads, etc.). This situation generates uncertainty, discouraging their implementations in real-world clouds. This paper introduces CloudBench, a methodology to facilitate the evaluation and deployment of VM placement strategies in private clouds. CloudBench considers the integration of a cloud simulator with a real-world private cloud. Two main tools were developed to support this methodology, a specialized multiresource cloud simulator (CloudBalanSim), which oversees evaluating VM placement techniques, and a distributed resource manager (Balancer), which deploys and tests in a real-world private cloud the best VM placement configurations that satisfied user requirements defined in the simulator. Both tools generate feedback information, from the evaluation scenarios and their obtained results, which is used as a learning asset to carry out intelligent and faster evaluations. The experiments implemented with the CloudBench methodology showed encouraging results as a new strategy to evaluate and deploy VM placement algorithms in the cloud.
**Keywords** Load balancing · Cloud simulator · Cloud resource management ·

## 1 Introduction

Cloud computing is a paradigm in which computing resources, such as computing units, storage, servers, applications, etc., are offered as an on-

demand services throughout the Internet [7, 13, 27]. Cloud resources are accessed without the need for users to be aware of their physical location and configurations. In the cloud computing paradigm, virtualized hardware, e.g., a virtual machine (VM), is provided following the infrastructure as a service (IaaS) model. In this model, VMs are instantiated in different physical machines (PMs) of the cloud infrastructure. The problem of deciding which PM will be the best option to host a VM is called the VM placement problem [8, 26]. This problem is a NP-hard problem [17] that has been formulated as the bin-packing problem [6], where analogously the VMs are packaged into PMs. To address this problem, exact and heuristic algorithms have been proposed. Exact algorithms guarantee delivering an optimal solution usually considering theoretical scenarios, whereas heuristic algorithms present practical procedures that do not offer such a guarantee but some feasible solutions, being the latter the majority of the current proposals [25].

Live migration plays a key role for the efficient placement of virtual machines (VMs) in data centers. It consists in moving an instantiated VM from one physical machine (PM) into another one, caused by a reaction to changes in the VM requirements or overloading problems in the hosting PM. Live migration is a process used by load balancing schemes to distribute resource consumption in the cloud. However, it usually implies a cost, e.g., it can be a resource and time-consuming process that can affect the fulfillment of the service level agreements (SLAs) [8]. In this context, VM placement strategies must control the number of live migrations [21], avoiding negative impact in the quality of the service. In a cloud infrastructure, load balancing algorithms are applied at two levels: at the application level and at the VM level. In the former, the load balancing algorithm is integrated into the application scheduler; while in the latter, it can be integrated into a VMs manager, which are the type of algorithms considered in the CloudBench proposal. Load balancing algorithms can also be centralized or distributed. In the centralized algorithms, one controller manages the entire system, which represents a single point of failure [22, 24, 36]. Distributed algorithms avoid this problem, but their complexity is higher due to the need of more coordination and control requirements. The cloud deployment models, public, private and hybrid, also present different considerations [36] that should be taken into account when analyzing a load balancing algorithm, for instance: (a) public clouds usually do not provide complete information over data, network and security settings and, in addition, their APIs change constantly due to the lack of standardization, which makes it difficult to capture all the information of the PMs and VMs; (b) private clouds are more suitable for evaluating VM load balancing strategies, since they allow users to include restrictions such as limiting the number of migrations and performance of

PMs; and (c) hybrid clouds can face communication restrictions and limitations defined by a specific data center and migration operations may require moving VMs from one cloud into another. The evaluation of different load balancing or VM placement strategies for an efficient deployment of VMs in clouds is not a trivial task [23, 24]. One of the difficulties in testing on real cloud environments is that it could be tedious, time-consuming and most of the time the cloud resources are not available for this purpose [4]. Most of the existing load balancing algorithms have been implemented in simulators, where there is usually a limited evidence that the proposed strategies can be implemented in real environments [29].

This paper presents CloudBench, a methodology for evaluating and deploying virtual machine placement strategies in an infrastructure as a service (IaaS) model in private clouds. Two supporting tools were developed for implementing the CloudBench methodology, a cloud simulator (CloudBalanSim) and a distributed resource manager (Balancer). CloudBalanSim allows users to evaluate different multi-resource load balancing strategies for VM placement in clouds according to specified performance metrics. Balancer is a distributed resource manager designed to test VM placement algorithms in a real-world private cloud, reproducing the test scenarios that showed satisfactory results during the CloudBalanSim simulation. In the CloudBalanSim methodology, the settings, algorithms, test scenarios and results generated during the simulation and real cloud tests are saved in a historical repository. This information becomes a learning asset to support the generation of more intelligent evaluations of VM placement algorithms. The major contribution of this work is:

1. A methodology (CloudBench) to improve cloud resources management, using an incremental learning approach, from evaluating different VM placement algorithms on a test infrastructure that combines a cloud simulator with a real cloud implementation. As a proof of concept, it was necessary to develop two tools, CloudBalanSim and Balancer, which are also contributions of this work.
2. An IaaS cloud simulator (CloudBalanSim) that allows the evaluation of different multi-resource strategies for VMs selection and placement, which is the only simulator of its kind that provides information of the following performance metrics: level of resource imbalance in the cloud (RAM and CPU), number of VM migrations and level of CPU performance degradation based on the Service Level Agreement Violation Time per Active Host (SLATAH) measurement [2].
3. A distributed and fault-tolerant cloud resource manager (Balancer), for testing and deploying VM placement algorithms in a real-world cloud.

4. Results of the evaluation of different state-of-the-art VM selection and placement strategies using CloudBench, identifying those whose results are very similar in both simulated and real clouds.

The rest of this document is organized as follows: Sect. 2 presents the related work, Sect. 3 describes the CloudBench methodology, Sect. 4 introduces CloudBalanSim and Balancer, which are the two main tools developed for supporting the CloudBench methodology, Sect. 5 presents the experiments and results obtained as a proof of concept of the CloudBench methodology, and finally in Sect. 6 the conclusions and future work are given.

## 2 Related work

This section presents relevant proposals that address the problem of load balancing in the Cloud, focusing mainly on two aspects: (a) solutions for the VM placement problem and (b) cloud simulators to evaluate different aspects of cloud resource management.

### 2.1 Virtual machine selection and placement for load balancing in the cloud

The VM placement problem is formulated as the bin-packing problem [6], in which objects of a given size must be packed in a minimum number of containers of a certain capacity [16]. In this analogy, the objects to be packed are the VMs, and the PMs represent the containers that have to be managed in the Cloud [25]. There are heuristics to solve the bin-packing problem, such as [21]: the First-Fit (FF), where each object is placed in the first container where it fits; the Best-Fit (BF), where each object is placed in the container where it fits and the rest of the remaining capacity is minimal; and the Worst-Fit (WF), where each object is placed in the container where it fits and the rest of the remaining capacity is maximum. Such heuristics can be improved if the objects are first sorted in decreasing order according to a determined weight, leading to modified algorithms such as the Best-Fit Decreasing (BFD). Sato et al. [26] considered the bin-packing formulation for the VM placing problem with the purpose of reducing the number of unnecessary live migrations. They proposed a dynamic optimization of the VMs placement by predicting the future resource usage through an auto-regressive model and solving the bin-packing problem through dynamic programming, based on the previous prediction of the resource usage. Its main objective is to manage the VMs in order to save energy and prevent the lack of resources.

Chen et al. [5] proposed a proactive load balancing model based on Markov decision processes (MDP) that serves to select the VM to be migrated and the destination PM, with the aim of reducing SLAs violations, overload and delay caused by load balancing. Kuo et al. [18] proposed an algorithm for VMs placement, the resource-based first-fit algorithm (RFFA), to find the first PM that met the resource constraints (CPU, memory, disk and network bandwidth) of a VM, to finally assign such VM to the PM. Beloglazov and Buyya [3] proposed and implemented an architecture for dynamic and energy-efficient VMs consolidation in OpenStack clouds, called OpenStack Neat. They addressed the VMs placement problem as the bin-packing problem for which they used a modified version of the BFD heuristic, in which the selection of the PM that would host the VM is done using BFD based on the CPU requirements and subsequently used the FF heuristic on the RAM requirements.

Most of these proposals were tested in simulation environments and some of them are theoretical solutions. Relevant cloud simulators used to analyze different aspects of cloud resource management are presented in the next section.

### 2.2 Cloud simulators

Currently, there exists an increasing interest of simulation tools that allow users the rapid development and evaluation of different strategies for VM placement, load balancing, VMs migration, among others [30]. There are different simulators that have been developed to perform experimentation in cloud environments. CloudSim [4] is one of the most popular [1, 37] and sophisticated [20] cloud simulation tool. It is a discrete event simulator implemented in Java that can be extended to incorporate new features. CloudSim allows the modeling and simulation of large-scale cloud computing environments, implements policies for overload detection, VMs selection and placement and allocation of resources (e.g., memory, processor, etc.). Despite the fact that CloudSim allows modeling the consumption of resources such as CPU, RAM and bandwidth, the consumption of a specific resource is analyzed in an isolated way, which means that the consumption of a combination of more than one resource is not considered [19].

NetworkCloudSim extends the CloudSim features to allow the modeling of more specialized and complex applications, e.g., a multi-layer web application, which consists of several layers, each one running on a different server with the possibility of communication among them. One of its main interests was to provide a network flow model that allows the creation of network topologies, integrating different types of switches, such as: RootSwitch, AggregateSwitch and EdgeSwitch [1, 11]. In a load balancing context, NetworkCloudSim does not support the simulation of VM live

migrations, which is an important limitation for VM placement strategies. MultiRECloudSim [19] is another simulator that extends CloudSim, adding multi-resource tasks (cloudlets) scheduling and an energy consumption model mainly based on CPU usage. MultiRECloudSim supports static and dynamic CPU workloads, whereas the other resources (RAM, I/O and bandwidth) use static workloads. It has special focus on task scheduling and energy saving, without carrying out load balancing actions either through tasks or VMs migration. CloudSched [31] is a simulator (implemented in Java) of IaaS clouds that takes into account multiple resources (CPU, RAM and bandwidth) in an integrated way for VMs scheduling. It models the VMs arrival process, the service time and the users requirements, randomly generating different VMs types with the required capacity. It also incorporates multi-resource metrics to measure the PMs average imbalance values. CloudSched does not support resource over-allocation (e.g., allocating more virtual CPUs—vCPUs—to VMs than the CPU cores available on the hosting PM), which is an aspect that fosters PM consolidation. It is not clear if CloudSched allows VM migrations, because there is not way to know how many migrations are executed when running a load balancing strategy.

DCSim (Data Center Simulator)[1] [32] is a simulator (implemented in Java) designed to study the VMs management in IaaS cloud data centers. It supports CPU over-allocation and models replicated VMs that share an incoming workload. DCSim models PMs power consumption based on CPU usage and allows VMs migrations. However, DCSim presents limitations when deciding VMs migration time, because it is calculated using the initial predefined memory capacities, considering only a dynamic behavior in the CPU.

FlexCloud[2] [34] is an open-source simulator (implemented in Java) of large-scale IaaS clouds that allows modeling the initialization process of clouds data centers, the assignment and migration of VMs and evaluating the performance of different load balancing algorithms and energy-efficient scheduling policies. The VM requests can be generated using the Poisson, Normal or Random distributions. However, it does not consider the time it takes to carry out the migration within the simulation time, nor does it take into account the VMs performance degradation. Like CloudSched, FlexCloud does not support over-allocation of resources.

As summary, we can say that CloudSim is a robust simulator that has influenced others but that it still presents limitations for the study of VM placement algorithms. It is also detected that none of the mentioned simulators supports the dynamic consumption of multiple resources, so their

[1] The DCSim code is public and can be downloaded from: https ://githu b.com/digs-uwo/dcsim .
[2] FlexCloud: https ://sourc eforg e.net/proje cts/flexc loud/.

VM placement strategies are limited to only considering: (a) the dynamic consumption of CPU, with static capacities for the rest of the resources with which VMs were instantiated; or (b) the static capacity with which VMs were instantiated for all resources. In this context, there still remains a need for an integrated evaluation tool that takes into account the dynamism of multiple resources, allowing users to experiment with VM placement and selection algorithms that provide load balancing in the cloud, using simulated and real clouds [19]. This situation motivated us to develop CloudBalanSim and Balancer tools, as part of the tools supporting the CloudBench methodology.

### 2.3 Discussion

To the best of our knowledge, there is not a similar methodology to compare with CloudBench. However, since CloudBench requires a cloud simulator and a tool to evaluate different VMs placement strategies in real clouds, we have focused the revision of the state-of-the-art on existing cloud simulators and VMs placement approaches. We realized that none of the existing work met the requirements of the CloudBench methodology, making necessary to develop a new cloud simulator (CloudBalanSim) and a distributed and fault-tolerant VMs deployment and load balancing tool (Balancer). Next, we summarize the main aspects that make different CloudBalansim and Balancer of existing proposals. Table 1 compares different VM placement proposals with our Balancer tool. We considered features such as the implemented approach, metrics, type of proposal and its objective. As we can see, Balancer is the only implemented tool for real-world clouds that allows evaluating and deployment of different VMs load balancing strategies (VM selection/placement and PM overload detection), offering fault-tolerance support (Sect. 4.2).

On the other hand, Table 2 summarizes the important simulation features that were required by CloudBench, such as: (1) Open Source, (2) VMs Migration, (3)

**Table 1** SummaryofVMplacementworks

| Work | Approach | Metrics | Proposal | Objective |
|---|---|---|---|---|
| Satoetal.[ 26] | Dynamicprogramming | CPU,RAM | Method | Toreduceunnecessarylivemigrations |
| Chenetal.[ 5] | Markovdecisionprocess | CPU,RAM | Model | ToreduceSLAsviolations,overloadanddelay |
| Kuoetal.[ 18] | Resource-basedFirst-Fit | CPU,RAM,disk, network | Algorithm | TofindthefirstPMthatmeettheresourceconstraints |
| BeloglazovandBuyya[ 3] | ModifiedBest-FitDecreasing | CPU,RAM | Implementation | Energy-efficientconsolidationofVMs |
| Balancer(ourproposal) | VMdeploymentandloadbalancing | CPU,RAM | Implementation | ToallowdifferentVMloadbalancingstrategieswith fault-tolerancesupportinreal-worldclouds |

**Table 2** Summaryofcloudsimulators

| Cloudsimulator | Opensource | VMsmigration | DynamicCPU consumption | DynamicRAM consumption | CPUover-allocation | Multi-resourceVM loadbalancing | Imbal-ance metrics |
|---|---|---|---|---|---|---|---|
| CloudSim[ 4] | ✓ | ✓ | ✓ | | ✓ | | |
| NetworkCloudSim[ 1, 11] | ✓ | | ✓ | | | | |
| MultiRECloudSim[ 19] | | | ✓ | | | | |
| CloudSched[ 31] | | | ✓ | | | | ✓ |
| DCSim[ 32] | ✓ | ✓ | ✓ | | ✓ | | |
| FlexCloud[ 34] | ✓ | | ✓ | | | | |
| CloudBalanSim(ourproposal) | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

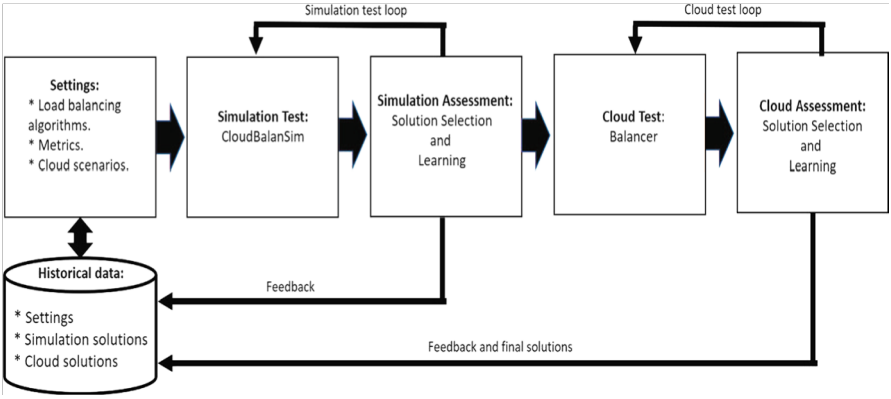



**Fig. 1** The CloudBench methodology

Dynamic CPU Consumption, (4) Dynamic RAM Consumption, (5) CPU Over-allocation, (6) Multi-resource VM Load Balancing, and (7) Imbalance Metrics. As we can see, CloudBalanSim is the only cloud simulator that fulfills most of the features. We are improving documentation, software and demonstrating its benefits before releasing the first open-source version. The complete features of CloudBalanSim are explained in Sect. 4.1.

## 3 The CloudBench methodology

CloudBench is a methodology that integrates simulation and real-world cloud scenarios for an intelligent evaluation, validation, and deployment of new or existing multiresource VM placement strategies in clouds. Figure 1 shows a conceptual model of this methodology. The stages that compose this methodology are described next.

*Settings* The definition of cloud evaluation scenarios, algorithms, metrics, user restrictions, workloads, among others is carried out in this stage. Some examples of metrics and user requirements are the workload imbalance level, maximum performance degradation, CPU or memory saturation thresholds and number of VM migrations. Details of these metrics are given in Sect. 4.1.3. The definition of the simulated cloud infrastructure includes the maximum number of physical (PMs) and virtual machines (VMs), the resource characteristics (memory, CPU, storage, etc.), workloads (synthetic or real) and the VM selection and placement strategies that will be evaluated.

*Simulation test* In this stage, a set of simulations are carried out taking as input the settings (configuration file) defined in previous stage. CloudBalanSim

(Sect. 4.1) is the simulator developed for supporting the CloudBench methodology. In every simulation test, the metrics and restrictions are the same, the only variant is the combination of the load balancing algorithms that will be evaluated. This means that for each combination of algorithms a test simulation is executed. Before executing a new simulation test, a query is sent to the historical repository to verify if there exists information about a previous execution of this test. If historical data exist, it is used to avoid the execution of a new simulation.

*Simulation assessment* The evaluation and analysis of obtained results in the simulation test is done in this stage. Feasible solutions, according to user restrictions, are selected. Most relevant information resulting from the simulation test is stored in a historical repository. This information will be used in a incremental learning process to improve future evaluations. Simulation test and simulation assessment stages will work in a loop until all of the required combinations of algorithms are evaluated in the simulation test.

*Cloud test* In this stage, a real cloud is deployed to reproduce those tests that were selected as feasible solutions in the simulation assessment stage. We have developed a tool called Balancer (Sect. 4.2), which is in charge of deploying the real cloud scenario using the OpenStack platform [9] and executing the VM selection and placement algorithms. Results obtained in the cloud scenario are sent to the next stage for evaluation.

*Cloud assessment* This stage has a similar function to the simulation assessment stage but considering real cloud test scenarios. All of the relevant information and feasible solutions resulting from the cloud test stage are sent to the historical repository that will be used in the incremental learning process.

### 3.1 Algorithm

The general steps of the CloudBench methodology are summarized in Algorithm 1. The set of combined algorithms that are chosen for the evaluation (Line 1) belongs to the following universe $VMsSelSts \times VMsPlaSts \times PMsOverDecSts$ . Where *VMsSelSts*, *VMsPlaSts*, and *PMsOverDecSts* identify the VM selection, VM placement and PM overload detection functions, currently available in the CloudBalanSim simulator. The configuration file (*CloudInf*) will have the definition of the cloud infrastructure, i.e., number of PMs, VMs, and characteristics of the resources (CPU, RAM, storage). Two main loops represent the core processes of the integrated (simulation and real

cloud) CloudBench methodology. In the simulation loop (Line 5), the CloudBalanSim simulator is used to execute every combination of algorithms (Line 11), taking into account the cloud infrastructure and settings. Before executing a specific simulation test, a query is sent to the history repository (Line 7) to verify if there exists historical information about this test. If the query response is true, results are obtained avoiding a new simulation execution (Line 8). Next, a results assessment is carried out (Line 13), where satisfactory (positive) and no satisfactory solutions are obtained (*SimFeasibleSols* and *SimDiscardedSols* respectively). In the real cloud loop (Line 21), the distributed cloud manager, Balancer, is used to deploy and execute (Line 27) every combination of algorithms that produced feasible solutions in the simulation stage. The real cloud scenario considers the same settings defined in the simulation. Like the simulation stage, before executing a real cloud evaluation, a query is sent to the historical repository to obtain previous results (Line 23), if any. An assessment of the real cloud results is made (Line 29), where satisfactory and no satisfactory solutions are obtained (*RCloudFeasibleSols* and *RCloudDiscardedSols* respectively). For learning reasons, at the end of the simulation (Line 18) and real cloud tests (Line 34), if historical information was not used (*previous* == *False*), satisfactory and no satisfactory solutions with their respective settings are saved into the historical repository (feedback).

**Algorithm 1:** CloudBench methodology.

**Input:** Algorithms (VMs Selection $VMsSelSts$, Placement $VMsPlaSts$, PMs Overload detection $PMsOverDecSts$)
**Output:** Feasible solutions in real cloud ($RCloudFeasibleSols$)

$CombinedStrategies \longleftarrow VMsSelSts \times VMsPlaSts \times PMsOverDecSts$
$SimFeasibleSols \longleftarrow \emptyset$; $SimDiscardedSols \longleftarrow \emptyset$
$RCloudFeasibleSols \longleftarrow \emptyset$; $RCloudDiscardedSols \longleftarrow \emptyset$
$Settings \longleftarrow$ Settings.definition(); $CloudInf \longleftarrow$ Cloud.infrastructure()
**for** $CurCombStr \in CombinedStrategies$ **do**
    Verify if a previous simulation exists
    **if** $SimHistory.check(CloudInf, Settings, CurCombStr) == True$ **then**
        $SimRes \longleftarrow$ SimHistory.getResults($CloudInf, Settings, CurCombStr$)
        $previous \longleftarrow$ True
    **else**
        $SimRes \longleftarrow$ Simulation.CloudBalanSimExec($CloudInf, Settings, CurCombStr$)
        $previous \longleftarrow$ False
    **if** $Simulation.Assessment(SimRes) == Positive$ **then**
        Simulation result is a feasible solution
        $SimFeasibleSols \longleftarrow SimFeasibleSols \cup SimRes$
    **else**
        $SimDiscardedSols \longleftarrow SimDiscardedSols \cup SimRes$
    **if** $previous == False$ **then**
        Save historical data for incremental learning
        $SimHistory \longleftarrow Feedback.store(SimFeasibleSols, SimDiscardedSols, CloudInf, Settings)$
**for** $CurCombStr \in SimFeasibleSols$ **do**
    Verify if a previous real cloud evaluation exists
    **if** $RCloudHistory.check(CloudInf, Settings, CurCombStr) == True$ **then**
        $RCloudRes \longleftarrow$ RCloudHistory.getResults($CloudInf, Settings, CurCombStr$)
        $previous \longleftarrow$ True
    **else**
        $RCloudRes \longleftarrow$ RealCloud.BalancerExecution($CloudInf, Settings, CurCombStr$)
        $previous \longleftarrow$ False
    **if** $RealCloud.Assessment(RealCloudResults) == Positive$ **then**
        Real Cloud result is a feasible solution
        $RCloudFeasibleSols \longleftarrow RCloudFeasibleSols \cup RCloudRes$
    **else**
        $RCloudDiscardedSols \longleftarrow RCloudDiscardedSols \cup RCloudRes$
    **if** $previous == False$ **then**
        Save historical data for incremental learning
        $RCloudHistory \longleftarrow Feedback.store(RCloudFeasibleSols, RCloudDiscardedSols, CloudInf, Settings)$
**return** ($RCloudFeasibleSols$)

## 4 CloudBench supporting tools

CloudBalanSim and Balancer are two tools that were developed to support the implementation of the CloudBench methodology. The following section describes these tools and the way they interact.

### 4.1 CloudBalanSim

CloudBalanSim is a cloud simulator based on CloudSim [4]. It was developed for supporting the CloudBench methodology. Its main purpose is to carry out simulations to evaluate resources management strategies in clouds that implement the IaaS model, with special support for the evaluation of VM placement and multi-resource (CPU and memory) load balancing strategies. Several metrics are incorporated to measure the workload imbalance that is generated on PMs that make up the cloud. These metrics allow to measure the impact that VM placement strategies have on resource consumption and the CPU quality of service (QoS) offered to users. The modular design of CloudBalanSim allows to improve its functionality incrementally. The following sections describe the architecture of CloudBalanSim, its functionality, and the different multi-resource VM selection and placement strategies that are currently implemented.

#### 4.1.1 Simulator architecture

Figure 2 shows the two layers, Cloud Services and Cloud Resources, of the original CloudSim's architecture, and the components that were added (dark gray module) and modified (light gray modules) to create this new simulator.

At the cloud resources level, the *Data Center* module is in charge of updating the CPU and RAM consumption every time VMs and PMs receive an event. This is different compared with CloudSim, which only updates the CPU usage. At cloud services level, the *VM Placement* module includes different VM selection and placement strategies that can be used by the *VM Provisioning* module, considering a specified cloud infrastructure. When VM placement strategies are carried out, the *VM Provisioning* module considers both the dynamic CPU and RAM consumption measurements. This information also provides a more realistic scenario for CloudBalanSim to carry out an efficient VM live migration. The different load balancing strategies and main metrics provided by CloudBalanSim are described in the following sections.

#### 4.1.2 Load balancing strategies

This section describes the VM selection and VM placement strategies with the PM overload detection functions implemented in the VM Placement module of CloudBalanSim.

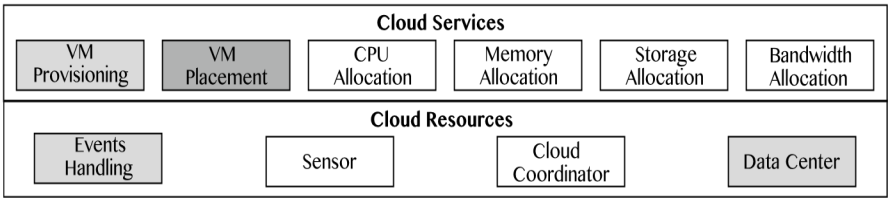



**Fig. 2** Cloud Services and Cloud Resources layers in CloudBalanSim

*VM selection strategies* These strategies are in charge of deciding which VM should be migrated from a source PM to a destination PM in order to release the excess of load in the source PM. We developed a new version of a VM selection strategy known as MMTMC [3], which considers the Minimum Migration Time and Maximum CPU utilization measurements. In this new version, called MMTMC2, the process of VM migration considers the amount of current RAM consumption, instead of using the total capacity with which the VM was originally instantiated, as is implemented in [3]. The VM with the shortest migration time (the least amount of RAM used) and with the highest CPU utilization will be chosen for migration in the PM or group of PMs that are overloaded (in one of their resources). This strategy is applied repeatedly in every PM until the overloading problem disappears. CloudBalanSim provides other 4 VM selection strategies that were inherited from CloudSim, [2] such as: MU (selects the VM with the lowest CPU usage); MMT (selects the VM with the least RAM capacity assigned at its creation time); RS (randomly selects the VM using a uniform distribution); and MCC (selects the VM with the maximum correlation coefficient between a list of VMs to be migrated).

*VM placement strategies* These strategies are in charge of selecting a destination PM, where the chosen VM (using a VM selection strategy) will be migrated, freeing resources of its current hosting PM (the overloaded PM). The current strategies implemented in CloudBalanSim are LIF, sandpiper, vectorDot, random and worstFit, which are explained below. These basic implementations do not consider any type of ordering based on the VM resource consumption. We also implemented a variant version of every strategy, where VMs are first ordered in a descending order, based on their RAM and CPU resource consumption, following that order, respectively. The corresponding version was identified adding the "Decreasing" word at the end of the name of the strategy: LIFDecreasing, sandpiperDecreasing, vectorDotDecreasing, randomDecreasing and worstFitDecreasing. For simplicity, in this section we describe only the basic implementations.

Worst-Fit In this placement strategy VMs are sequentially assigned into PMs that have available resources to host them. Each PM $i$ has a weight ( $weight_i \leftarrow \sum_{j=1}^{MWM.size()} MWM[j] \times normPM_{ij}$ ) calculated from the aggregation of available resources (previously normalized) multiplied by a weight factor that determines the relevance of each metric. The PMs that will be taken first are those with the greatest weight (the maximum amount of resources to offer).

Random Assigns the VMs sequentially into the randomly selected PMs that have the available resources for each metric, using a uniform distribution.

Sandpiper Captures the combined CPU, network and memory load of both a VM and a PM. Defines the volume of a VM or PM as the product of its CPU, network and memory loads ( $Vol = \frac{1}{1-cpu} \times \frac{1}{1-net} \times \frac{1}{1-mem}$ ); and in the case that some resource is being used completely (e.g., cpu=1 ), its consumption is set to $1-\epsilon$, instead of one, to avoid infinite volumes [33]. In our implementation, this metric was used to place the VMs sequentially into the PM that has sufficient resources for each metric, and with the lowest volume. The network resource was not taken into account when calculating Vol, so its value will be $Vol \leftarrow \frac{1}{1-cpuU(PM_i)} \times \frac{1}{1-memU(PM_i)}$, where $cpuU(PM_i)$ and $memU(PM_i)$ represent the CPU and RAM utilization percentage, respectively, which are in the range $[0,1-\epsilon]$.

VectorDot In this placement strategy, VMs are sequentially allocated into the PM that has sufficient resources for each metric and that, in turn, has the lowest value obtained using the dot product between the load fractions vector VVec and the smoothing of the load fractions vector LVec with respect to its thresholds vector TVec ( $val = \sum_{i=1}^{LVec.size()} VVec[i] \times \exp(\alpha \times \frac{LVec[i]-TVec[i]}{TVec[i]})$ ), where $\alpha$ is a smoothing constant [28]. In our implementation, VVec represents the fractions vector of the amount of required resources for each VM metrics and the amount of available resources for each PM metrics. LVec represents the resources usage percentage from each PM metrics and TVec represents the threshold established

for each PM metrics. Only the CPU and RAM metrics were considered, and the constant was established to $\alpha$=1 to give all metrics the same importance.

LIF The Lowest Integrated-load First (LIF) strategy assigns the VMs sequentially to the PM with the lowest integrated load. Based on the VM requirement characteristics (e.g., CPU vC, memory vM, network bandwidth vN, etc.), LIF always selects the PMs with the lowest integrated load ( $Avg_i \leftarrow \frac{CPU_{Ui}+MEM_{3\,Ui}+NET_{Ui}}{}$ ) that have sufficient resources for each metric to assign the VMs [38]. In our implementation, the network resource was not taken into account when calculating $Avg_i$ , so it would be: $Avg_i \leftarrow \frac{CPU_{Ui}+2MEM_{Ui}}{}$.

*PM overload detection functions* These CloudBalanSim functions have as main aim to detect when any of the PMs in the cloud is receiving an excess of load in one or more of its resources, avoiding or foreseeing performance degradation problems. These functions complement the VM placement strategies. Basic PM overload detection functions allow to define static thresholds (THR) for CPU and RAM consumption in every PM. Equation 1 defines a simple function to verify these thresholds with respect to current consumption. Where $t_k$ represents the current time; $U_{CPU}(PM,t_k)$ is the percentage of CPU usage of the PM at time $t_k$ ; $U_{RAM}(PM,t_k)$ is the percentage of RAM usage of the PM at time $t_k$ ; $T_{CPU}$ is the CPU usage overload threshold; and $T_{RAM}$ is the RAM usage overload threshold.

$$overload(PM,t_k) = \begin{cases} true & \text{if } U_{CPU}(PM,t_k) > T_{CPU} \text{ or} \\ & U_{RAM}(PM,t_k) > T_{RAM} \\ false & \text{otherwise} \end{cases} \quad (1)$$

Other PM overload detection functions allow the detection of CPU over-utilization using dynamic load threshold for CPU based on metrics such as the mean absolute deviation (MAD) or the interquartile range (IQR). Some

functions predict CPU utilization and detect potential overloading using local regression (LR) or local robust regression (LRR) techniques [2].

Finally, Table 3 summarizes all the aforementioned VM selection, VM placement and PM overload detection strategies, which in turn were implemented in CloudBalanSim.

#### 4.1.3 Performance metrics

CloudBalanSim provides specific metrics to evaluate load balancing in the cloud during the execution of VM placement strategies. Cloud imbalance level, percentage of service level agreement (SLA) violations and number of VM live migrations are examples of these metrics.

*SLA violation time per active host (SLATAH)* Service level agreement (SLA) fulfillment involves different aspects of the cloud infrastructure. However, currently implementation of CloudBalanSim only defines SLA violations in terms of CPU degradation in physical machines (PMs). To measure the CPU SLA fulfillment, the SLA Violation Time per Active Host (SLATAH) [2] metric is used. This metric shows the percentage of time active PMs have experienced 100% CPU utilization (Eq. 2). The reasoning of the SLATAH metric is that if a CPU of a PM that is hosting VMs is experiencing 100% of utilization, the SLA established for the VMs, in terms of performance, could be violated.

$$\text{SLATAH} = N1^{-} \sum_{i=N1} TTas_{ii} \tag{2}$$

In Eq. 2, $N$ is the number of PMs; $T_{si}$ is the total time during which the PM $i$ has experienced 100% CPU usage leading to a SLA violation; and $T_{ai}$ is the total time that the PM $i$ has remained in an active state (hosting VMs).

*Cloud imbalance level* CloudBalanSim provides two metrics for measuring the load imbalance level of the cloud, considering the RAM and CPU consumption in all of the PMs: $\text{IBL}^{\text{CDC}}{}_{\text{avg}}$ [31, 35, 38] and *TotalIBScore* [28]. These metrics allow detecting if the RAM and CPU consumption (load) is disproportionate in the cloud infrastructure. The definitions of these metrics are the following:

– $\text{IBL}^{\text{CDC}}{}_{\text{avg}}$ : It measures the average imbalance value by calculating an aggregate of the load variances of each resource in the cloud.

$$IBL_{avg}^{CDC} = \frac{\sum_{i=1}^{N} (RAM_{Ui} - RAM_{Au})^2 + \sum_{i=1}^{N} (CPU_{Ui} - CPU_{Au})^2}{N} \quad (3)$$

In Eq. 3 $RAM^{U}_{i}$ and $CPU^{U}_{i}$ are the average RAM and CPU utilization of a PM $i$. $RAM^{A}_{u}$ and $CPU^{A}_{u}$ represent the average utilization of all the RAM and CPUs of the cloud, respectively, and $N$ the total number of PMs in the cloud.

– *TotalIBScore*: It is the measure of the total imbalance in the cloud with respect to a defined threshold. The total imbalance score is calculated by aggregating the imbalance score of each node in the cloud. Equation 4 shows such metric.

$$TotalIBScore = \sum_{i}^{u} IBscore(u)$$

$$IBscore(u) = IBscore(NLFVec_i(u), NTVec_i(u))$$

$$IBscore(f,T) = \begin{cases} 0 & \text{if } f < T \\ \exp\left(\frac{f-T}{T}\right) & \text{otherwise} \end{cases} \quad (4)$$

Where $NLFVec_i(u)$ is the i-th element of the load fraction vector calculated by dividing the resource consumption by the capacities of the PM $u$. $NTVec_i(u)$ is the i-th element of the thresholds vector of the PM $u$, and $IBscore(f,T)$ is an exponential weighting function, where $f$ is the resource's load fraction and $T$ is the corresponding threshold.

The objective of a load balancing strategy is to reduce the value of the imbalance metrics as much as possible by migrating VMs [28].

*Number of migrations* The number of VM migrations is a metric that can be used in a complementary way to measure the load balancing. It is advisable to use this metric as a complement to others, since using it as the only reference metric to evaluate the load balancing effects can produce undesirable results. For example, a high number of VM migrations could lead the cloud to a state of balanced loads, but they could also cause a significant performance degradation.

### 4.2 Balancer

This section describes the architecture and implementation of Balancer, a distributed and fault-tolerant resource manager for virtual machine (VM) deployment and load balancing in the cloud. Balancer is the component of CloudBench that allows to reproduce and verify, in a real-world cloud, the simulations and results obtained by CloudBalanSim. The current implementation of Balancer is developed on the OpenStack platform [9].

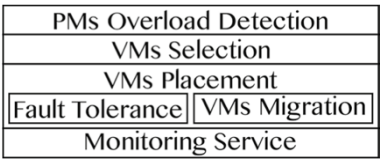



**Fig. 3** Balancer architecture

#### 4.2.1 Architecture

Balancer has a modular architecture (see Fig. 3) mainly composed of the following components: Physical machine (PM) overload detection, VM selection, VM placement and monitoring service. These modules work as distributed services in the cloud nodes. Scalability and fault tolerance are two non-functional aspects included in Balancer. The following sections describe the most important Balancer modules.

*Monitoring* This service is in charge of gathering performance metrics from both the physical and virtual cloud infrastructure. Monitoring provides timely information that improves the decision-making process during the execution of the load balancing strategies. This service is used by all modules of Balancer. Current implementation of Balancer integrates the Monasca [3] monitor, which provides monitoring and logging as-a-service for the OpenStack platform. The Monasca monitoring service was modified so that, in addition to sending the data to a central database of metrics, each node can store its own metrics locally in a time series database (InfluxDB). In this way, Balancer keeps the monitoring data distributed to facilitate its use to the load balancing strategies. Monasca is a well-supported monitor that has showed good performance in very overloaded environments [12].

[3] http://monas ca.io/about .html, https ://wiki.opens tack.org/wiki/Monas ca.

**Table 3** Summary of load balancing strategies

| VM selection strategies | VM placement strategies | PM overload detection functions |
|---|---|---|
| MMTMC, MMTMC2, MU, MMT, RS, and MCC | LIF, LIF Decreasing, sandpiper, sandpiper Decreasing, vectorDot, vectorDot Decreasing, random, randomDecreasing, worstFit, and worstFit Decreasing | Multi-resource: static-multi-resource threshold. Single-resource: static threshold, MAD, IQR, LR and LRR |

*PM overload detection* This module is responsible for the overload detection of PMs, implementing the same strategies used in CloudBalanSim, as were explained in Sect. 4.1.

*VM selection* It is in charge of selecting the set of VMs to be migrated from overloaded PMs to PMs with more available resources. The following VM selection strategies are currently part of this module: MMTMC2, RS and MU; explained in Sect. 4.1.2. However, it is possible to add all the VM strategies already implemented in CloudBalanSim (Sect. 4.1). It is worth mentioning that when Balancer is working as part of the CloudBench method (Sect. 3), the VM selection module will use only the VM selection strategies that showed satisfactory results in the simulation test (CloudBalanSim).

*VM placement* The VMs that were chosen by the VM selection module represent the input for the VM placement module, which is in charge of finding the best possible destinations (PMs) for executing VM live migrations. Different VM placement strategies were implemented in this module. However, as occurs in the module of VM selection, Balancer will use only the strategies that showed satisfactory results in CloudBalanSim if it is running as part of the CloudBench method. The current implementation of this module includes the following VM placement strategies: WorstFitDecreasing, Random, Sandpiper, VectorDot and LIF (Sect. 4.1.2). However, it is possible to integrate all of the VM placement strategies included in CloudBalanSim. The metrics used in CloudBalansim to measure cloud load balancing and quality of service (Sect. 4.1.3) are also supported by Balancer.

#### 4.2.2 Fault tolerance

The VM placement module needs to maintain a global view of the information of all cloud nodes (PMs) that will be managed. This requirement led us to a centralized design, limiting fault tolerance and scalability. To address this issue, we designed a distributed VM placement module that provides fault-tolerance and scalability support. In the distributed version, a new algorithm for choosing a coordinator (or leader) node was implemented. This algorithm, named Dynamically Weighted Bully (DWB), is a variant of the Bully algorithm [10]. Unlike the original Bully algorithm, where the selection of a coordinator considers a static identification of the nodes, in the DWB algorithm a weight dynamically assigned to the node is used as identifier. The weight also indicates the amount of available resources in each node. The node with the greatest weight is the node that is chosen as the coordinator. In this way the coordinator selection follows a similar technique to the Worst-

Fit strategy (Sect. 4.1.2). The coordinator will be in charge of finding a destination PM to a VM chosen for migration. If the coordinator fails, another is chosen using the DWB algorithm.

### 4.2.3 Scalability

In the physical cloud infrastructure, Balancer allows to create groups of nodes (PMs) to be managed independently. The cloud infrastructure can be divided into subgroups of nodes, where every subgroup can choose a coordinator (or leader). This group design avoids concentrating the VM placement process in only one coordinator node. If a group coordinator cannot find an appropriate PM to place the VMs, Balancer can be extended to redirect VMs migration requests to a coordinator of another group, performing inter-group load balancing (see Fig. 4). This design facilitates Balancer to scale.

### 4.2.4 Balancer integration with OpenStack

Balancer was designed to be integrated into the OpenStack[4] cloud platform, one of the most popular tools used to build public, private and hybrid clouds [14–16].

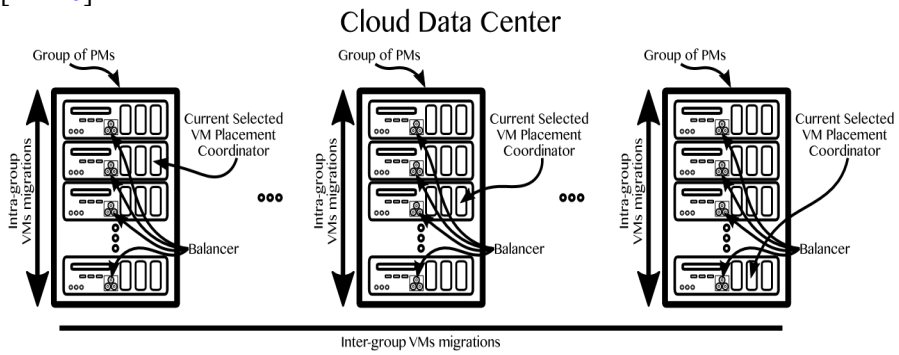



**Fig. 4** Scalability of the VMs placement module

Balancer's modules can be installed as additional OpenStack services (e.g., novacompute). In this way, when an OpenStack cloud is used, it is not necessary to modify the cloud environment or make specialized

[4] https ://www.opens tack.org.

configurations of it, taking advantage of existing OpenStack functional services, such as VMs live migration, infrastructure monitoring, etc.

## 5 CloudBench: experimental evaluation and results

This section describes the experiments carried out to evaluate different proposals of VM selection and placement strategies to provide load balancing in clouds, using the CloudBench methodology. The obtained results are also shown. According to the recommendations of the CloudBench methodology, the experiments were executed and evaluated on simulated and real-world cloud scenarios. CloudBalanSim was used in the simulated scenario, whereas Balancer was used in the real-world cloud scenario. It is worth mentioning that sudden load peaks for resource demands in VM or PM were not considered in these experiments, as they happened in negligible periods of time compared to the complete workload. These peaks do not represent a consistent overload; therefore, executing VM migrations based on this information could be costly. The analysis of this behavior is beyond the scope of this article and is left as future work.

### 5.1 Simulation settings and test with CloudBalanSim

This section specifies the different scenarios to evaluate in the simulated environment, such as the VMs selection and placement strategies, the performance metrics, the cloud infrastructure to be simulated, as well as the workloads to be simulated.

#### 5.1.1 VMs selection-placement strategies and performance metrics

A total of 5×10×2=100 tests were executed, which result from the combination of the 5 VM selection strategies (MMTMC2, MU, MMT, RS, MCC) with the 10 VM placement strategies (LIF, LIFDecreasing, sandpiper, sandpiperDecreasing, vectorDot, vectorDotDecreasing, random, randomDecreasing, worstFit and worstFitDecreasing) described in Sect. 4.1.2, running on 2 cloud infrastructures, one with 41 physical machines (PMs) and another with 50 PMs. Different cloud scenarios with different number of PMs were tested. However, in this paper, we present the two scenarios with the most representative results. The PM overload detection function applied in these experiments uses a static threshold (THR) (Sect. 4.1.2), with RAM and CPU consumption thresholds set to 0.8. The following metrics (described in Sect. 4.1.3) were considered in these experiments: (1) $IBL^{CDC}_{avg}$ and *TotalIBScore*, which measure the level of imbalance in resource consumption generated by each combinations of strategies; and (2) the

number of VM migrations and SLATAH, which measure both the load balancing and the SLA violations in terms of CPU performance degradation.

### 5.1.2 Cloud infrastructure

This section describes the infrastructure simulated with CloudBalanSim. In all of the experiments, the same cloud infrastructure and workloads were used, making the comparison under the same conditions.

*PMs characteristics* Two types of PMs were simulated, their characteristics are shown in Table 4. The processor frequency of each PM was mapped to MIPS (Millions of Instructions Per Second). The server model and the number of PMs created are shown in Table 5.

*VMs characteristics* Table 6 shows the characteristics of the simulated VMs. Four different types of VM were defined for these experiments, varying the speed of the cores and the capacity of RAM. Every VM instance type was identified by a name (first column of Table 6). All VMs are single-core in accordance with the CPU usage traces provided by the workload applied in these experiments (see details in Sect. 5.1.3). Since at the beginning of the simulation, the VM resource consumption is not known, at the time of the VM instantiation the resource requirements are those predefined by the assigned VM instance type. However, the VMs can use a smaller amount of resources, depending on the resource consumption generated by the workload. The number of VMs created was 263 for each of the 4 types showed in Table 6, producing a total of 1052 VMs created per test configuration. This is the number of VMs required by the workload explained in the next section.

**Table 4** PMs characteristics

| | PM 1 | PM 2 |
|---|---|---|
| Server | Huawei RH2288H V2 | DEPO Race X340H |
| Processor | Intel Xeon E5-2609 | Intel Core i5-4570 |
| Cores | 8 | 4 |
| MIPS | 2400 | 3200 |
| RAM (GB) | 48 | 16 |
| Storage (GB) | 1000 | 1000 |

**Table 5** Model and number of servers created

| Server model | PMs | |
|---|---|---|
| Huawei RH2288H V2 | 21 | 25 |
| DEPO Race X340H | 20 | 25 |
| Total number of servers in the cloud | 41 | 50 |

**Table 6** VMs characteristics [2]

| VM instance type | Cores | MIPS | RAM (GB) | Storage (GB) |
|---|---|---|---|---|
| High-CPU medium | 1 | 2500 | 0.85 | 2.5 |
| Extra large | 1 | 2000 | 1.7 | 2.5 |
| Small | 1 | 1000 | 1.7 | 2.5 |
| Micro | 1 | 500 | 0.6 | 2.5 |

### 5.1.3 Workloads

CloudBalanSim allows to simulate both RAM and CPU consumptions, either by using workload traces (real or synthetic) or using a certain distribution. In each of the experiments the same workload was generated during 24 h. The VMs CPU consumption was modeled based on the real CPU usage traces of the PlanetLab network.[5] These traces were provided as part of the CoMon project, a PlanetLab's monitoring service. The total set consists of 10 days of collected traces, at a 5 min monitoring interval during periods of 24 h, between the months of March and April of 2011 [2]. For our tests, we selected the set of traces of the first day (03/03/2011), which consists of the CPU usage traces of 1052 VMs that were monitored that day. This workload requires CloudBalanSim to instantiate 1052 VMs, using a CPU overallocation level ≤ 6 (i.e., up to 6 virtual CPUs for each real CPU). The RAM consumption was modeled by a normal distribution with mean 0.8 and standard deviation 0.2. RAM consumption must be in the [0, 1] interval; so, in case of $RAM_U > 1$ it will take the value of 1.

[5] https ://githu b.com/belog lazov /plane tlab-workl oad-trace s.

**Fig. 5** CloudBalanSim: average of the $IBL^{CDC}_{avg}$ imbalance metric with **a** 41 PMs and **b** 50 PMs

### 5.1.4 Simulation test and assessment

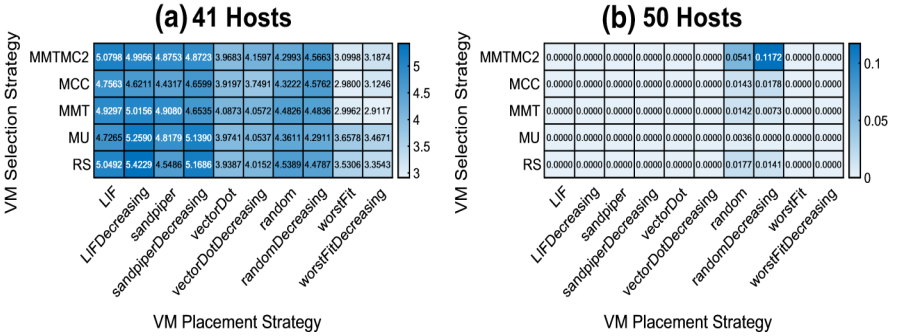
The simulation tests were mainly focused on two aspects: (a) validation of the CloudBalanSim functionality, and (b) evaluation and comparison of different VM selection and placement strategies for load balancing in the cloud. A successful execution of CloudBalanSim validated the first aspect. The second aspect determined which of the strategies offers the best load balancing level in the cloud.

Figure 5 shows a summary of the average imbalance level, $IBL^{CDC}_{avg}$, generated by each combination of the VM selection and placement strategies after executing in two cloud infrastructures, one with 41 PMs and another with 50 PMs. A lower value in $IBL^{CDC}_{avg}$ means a better load balancing in the cloud. It can be seen in Fig. 5a that with a high demand of CPU over-allocation in PMs (less PMs host more VMs), the combination of strategies that generated a lower level of imbalance (0.0077) was Sandpiper and MCC. If the number of PMs in the cloud is increased (Fig. 5b), the combinations of VM placement strategies that generated the least imbalance (0.0077) were LIF, LIFDecreasing, Sandpiper and SandpiperDecreasing, independently of the VMs selection strategy that was used. The rest of the VM placement strategies generated a greater imbalance. These tests show us that in scenarios with high demand for computing resources (high demand of CPU over-allocation), the VMs selection and placement strategies that present a better performance in terms of load balancing are Sandpiper and MCC, respectively. While in scenarios with a low demand of CPU over-allocation (the cloud has more available PMs), the VM placement strategies that performed best were LIF and sandpiper, both in their normal and decreasing versions.

We also analyzed the average of the *TotalIBScore* metric, which reflects the total cloud imbalance relative to a threshold. In this case, with a smaller number of PMs (Fig. 6a) the worstFitDecreasing and MMT strategies generated the least imbalance (2.91). By further increasing the number of PMs (Fig. 6b), the random and randomDecreasing strategies in combination with the majority of the VM selection strategies (except when randomDecreasing is combined with MU) produced an imbalance level greater than 0, while for the other strategies the level of imbalance was equal to 0. This metric shows that the Random strategies are more likely to exceed the PM overload detection threshold, while in general the vectorDot and worstFit strategies, both in their normal and decreasing versions, converge faster to a state of equilibrium, where the thresholds are not exceeded.

**Fig. 6** CloudBalanSim: average of the *TotalIBScore* imbalance metric with **a** 41 PMs and **b** 50 PMs

The number of VM migrations generated by each combination of strategies was also measured for the two cloud infrastructures. It was observed that with a high demand of CPU over-allocation in PMs (Fig. 7a), the WorstFit and RS strategies generated the lowest number of migrations (1199). By further increasing the number of PMs (Fig. 7b) the LIF and Sandpiper strategies, both in their normal and decreasing versions, converged to an optimal state, where the migration of VMs was no longer required. Considering the $IBL^{CDC}_{avg}$ imbalance level shown in Fig. 5b, where the same strategies had the lowest level, it is clear that a lower imbalance level generates a smaller number of




migrations. Something to note is that while the vectorDot, vectorDotDecreasing, worstFit and worstFitDecreasing strategies seem to converge more slowly toward a balanced state, random strategies keep resources in a constant imbalance state and consequently generate a greater

number of migrations, which could negatively impact performance in data transmission in the cloud.

The SLA fulfillment with respect to CPU performance in PMs is an aspect that can be analyzed with the SLATAH metric (Sect. 4.1.3). It was observed that when the number of PMs is reduced and there is a higher CPU over-allocation (Fig. 8a), the worstFitDecreasing and MU strategies generated the lowest SLATAH percentage (0.151%). By increasing the number of PMs (Fig. 8b), the LIF and Sandpiper strategies converged to the optimum where there was not CPU performance degradation. This occurred in both their normal and decreasing versions, independently of the VMs selection strategy. Considering the $IBL^{CDC}_{avg}$ imbalance level shown in Fig. 5b and the number of migrations in Fig. 7b, where LIF and Sandpiper strategies had the lowest level, we can see they favor a better use of the cloud resources in scenarios where the over-allocation of CPU is used. LIF and Sandpiper also kept a low level of imbalance and migrations without causing CPU SLA violations.

Table 7 summarizes the CPUs allocation in the two cloud scenarios (with 41 and 50 PMs). The second column shows the total number of cores in the PMs, considering the characteristics of the PMs shown in Table 4 and the number and models of server (PMs) described in Table 5. The third column shows the

**Fig. 7** CloudBalanSim: number of VM migrations with **a** 41 PMs and **b** 50 PMs

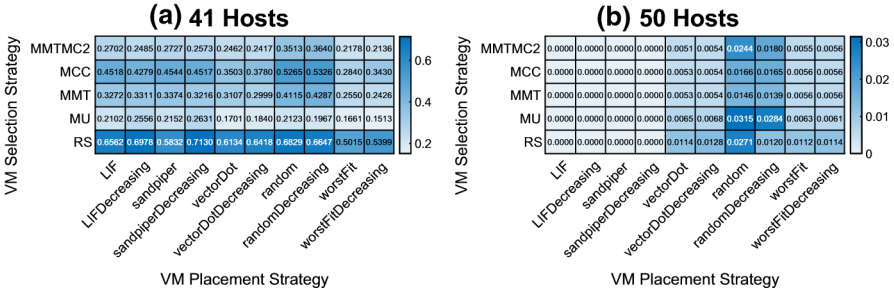
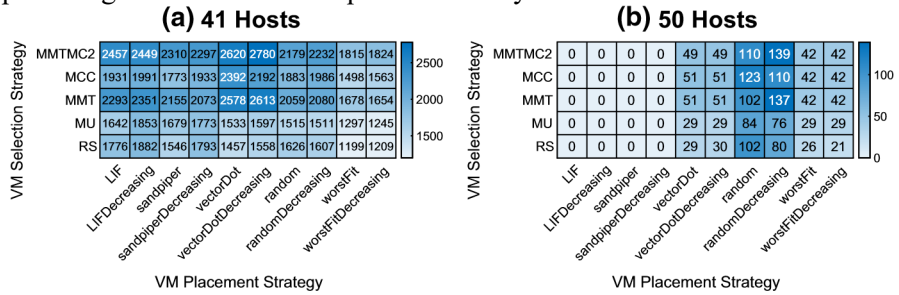



**Fig. 8** CloudBalanSim: SLATAH metric with **a** 41 PMs and **b** 50 PMs

percentage of cloud CPUs required to satisfy the allocation of the 1052 VMs




demanded in the workload. Recall that it was considered a maximum CPU overallocation level of 6, i.e., up to 6 VCPUs for each real CPU. A high value in this column, e.g., 100%, would mean that the VMs demanded 1 real CPU for each VCPU, whereas a low value means that a reduced number of real CPUs could host a higher number of VCPUs. The column "% CPUs savings" shows the complement of the previous values and represents the percentage of CPUs that remain available in the cloud. We can see that PMs consolidation produces more CPU or PMs savings (with 41 PMs, 76.43% of gain). As the number of PMs increases PM consolidation decreases. Depending of the user interest, the combination of VM selection-placement strategies can be configured for obtaining high PMs consolidation (which could be an energy saving approach) or an adequate load balancing (focused on better performance) or guaranteeing a low number of SLAs violations (quality of service).

### 5.2 Real cloud settings and test with balancer

According to the CloudBench methodology shown in Fig. 1, after the simulation stage, we continue with the real cloud evaluation, where Balancer comes into play. In this stage, we executed and validated the combination of VM selection and placement algorithms that showed satisfactory results in the simulation stage, using CloudBalanSim.

**Table 7 CPU** savings

| PMs | CPUs (total no. of cores) | % CPU allocated | % CPUs savings |
|---|---|---|---|
| 41 | $21 \times 8 + 20 \times 4 = 248$ | $\frac{248}{1052} \times 100 = 23.57$ | 76.43 |
| 50 | $25 \times 8 + 25 \times 4 = 300$ | $\frac{300}{1052} \times 100 = 28.51$ | 71.49 |

### 5.2.1 VMs selection-placement strategies and performance metrics

The CloudBalanSim evaluation produced fifteen strategies with satisfactory results. These strategies are the combination of the next 3 VM selection algorithms: MMTMC2, MU, RS; with the following 5 VM placement algorithms: LIF, sandpiper, vectorDot, random and worstFitDecreasing. For comparison reasons, it was also included an additional test used as baseline, in which none VM selection and placement strategy is executed, giving a total of 16 configuration tests. We also used the same PM overload detection function based on static threshold (THR) with RAM and CPU thresholds set to 0.8. As with the experiments carried out with CloudBalanSim, in Balancer the benefits were valued in terms of the load balancing level achieved from the CPU and RAM consumption in the PMs that make up the cloud. The metrics considered in these experiments were: $IBL^{CDC}_{avg}$ , *TotalIBScore*, number of VM migrations and SLATAH, which are the same metrics used in CloudBalanSim.

### 5.2.2 Cloud infrastructure

A private cloud was built using the OpenStack cloud platform, in which the Balancer tool was integrated to deploy and manage VMs. Different private cloud scenarios were generated to run the same evaluations. Since most of them showed a similar behavior, this paper presents a representative scenario.

*PMs characteristics* The private cloud was built with 5 OpenStack nodes (PMs), 1 controller and 4 compute, whose characteristics are shown in Table 8.

*VMs characteristics* Every VM instantiated in the cloud consists of 1 core, 2 GB of RAM and 10 GB of storage capacity. As with the tests performed with CloudBalanSim, all VMs are single-core in accordance with the PlanetLab workload specification. Table 9 shows the number of VMs created in every involved PM. A total of 168 VMs were created for each test configurations.

| PM | Cores | RAM (GB) | Storage (GB) | Frequency (GHz) |
|---|---|---|---|---|
| controller | 6 | 32 | 754 | 2.00 |
| compute9 | 12 | 64 | 488 | 2.20 |
| compute10 | 24 | 256 | 488 | 2.20 |
| compute11 | 24 | 128 | 488 | 2.20 |
| compute12 | 24 | 128 | 488 | 2.20 |

**Table 8** Hardware characteristics of cloud PMs

| PM | # VMs |
|---|---|
| Compute9 | 24 |
| Compute10 | 48 |
| Compute11 | 48 |
| Compute12 | 48 |

**Table 9** Number of VMs created per PM

### 5.2.1 Workloads

RAM and CPU consumption were generated by the same workloads used in CloudBalanSim. CPU consumption was generated by real CPU usage traces, taken from the PlanetLab network (see Sect. 5.1.3). RAM consumption was produced by a workload with synthetic traces. The RAM consumption follows a uniform distribution of random values between 0 MB and 1424 MB. The maximum value (1424 GB) was selected to avoid exceeding the VM RAM capacity (2 GB). The rest of memory was occupied by the operating system (Centos 7 x 86-64 bits).

### 5.2.2 Cloud test and assessment

This section shows the obtained results of two aspects evaluated in Balancer: (a) the functionality of the fault-tolerance scheme and (b) the validation of the VM selection and placement algorithms chosen by CloudBalanSim.

*Fault tolerance* To check functionality of the fault tolerance scheme of Balancer, the following test was performed. Balancer was executed on the OpenStack compute nodes described in Table 8. A turn-off process was activated in the compute10, compute12 and compute11 nodes, in sequential intervals of 10 min, to simulate failures in the nodes in that order. Later, a turn-on process was activated in the same nodes, but in reverse order. Figure 9 shows 2 values on the *y*-axes (Node and Weight). Node identifies a node of interest and Weight represents the amount of available resources in that node (higher value means higher availability). A green circle indicates the node that was chosen as the coordinator (Coord) in a determined time. A red circle indicates the node that is turned off (Down), simulating a failure. The value of
Weight is represented by bars. At the beginning of the test (min 0), the compute10

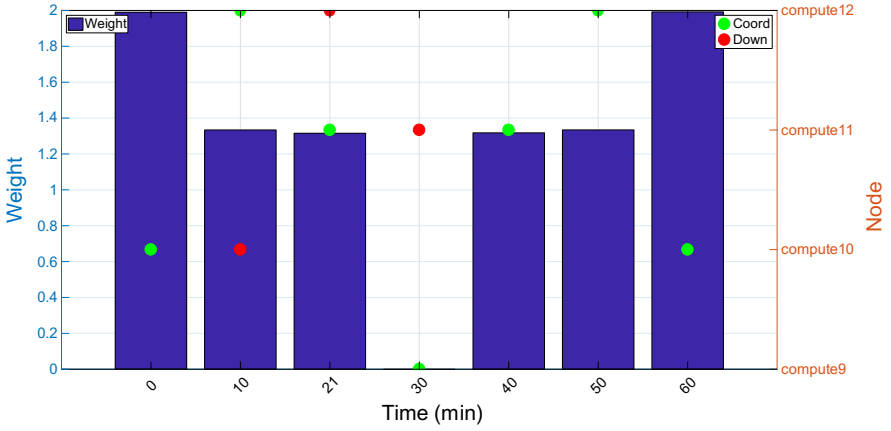



**Fig. 9** Balancer: fault tolerance

node (weight of 2) was chosen as coordinator; 10 min later, after failing compute10, the compute12 node (weight of 1.35) was elected as coordinator. 10 min later, after failing compute12, the compute11 node (weight of 1.3) was elected as coordinator; and 10 min later, compute11 was turned off and the compute9 node (weight of 0) was elected as coordinator. It is possible to appreciate the downward trend of the weight as the failures were presented

in the coordinators, always choosing as the new coordinator the active node with greater weight. Afterward, when the nodes were turned on again in reverse order, an upward trend can be seen reaffirming that the active node with the greatest weight is always chosen as the new coordinator.

*Comparison of the VM selection-placement strategies* Figure 10a shows the results obtained for every VM selection-placement strategy in terms of the average imbalance level metric, $IBL^{CDC}_{avg}$ . It can be observed that LIF and MMTMC2 were the combination of algorithms that generated a lower imbalance level (0.0257) and that the use of the MU algorithm always generated a higher imbalance level. If we also analyze the average of the *TotalIBScore* metric, we see (Fig. 10b) that the worstFitDecreasing and MMTMC2 strategies generated the least imbalance (1.6). In general, all strategies had a very similar imbalance level, and in none of the cases a value of 0 was obtained.

The number of VM migrations generated in each evaluation was also measured (Fig. 11a). It can be seen that the worstFitDecreasing and MMTMC2 strategies generated the lowest number of migrations (218), whereas random and MU generated the highest number of migrations (257). The combination of strategies that generated a lower imbalance level $IBL^{CDC}_{avg}$ (LIF and MMTMC2) is one of those that generated a greater number of migrations (246). In this case, it was possible to execute a greater number of migrations to maintain equilibrium. However, this was not possible for other strategies due to the restriction of not overloading the destination PM.

The SLA fulfillment in terms of CPU degradation was measured using the SLATAH metric. Firstly, we executed our baseline test, which does not include the VM

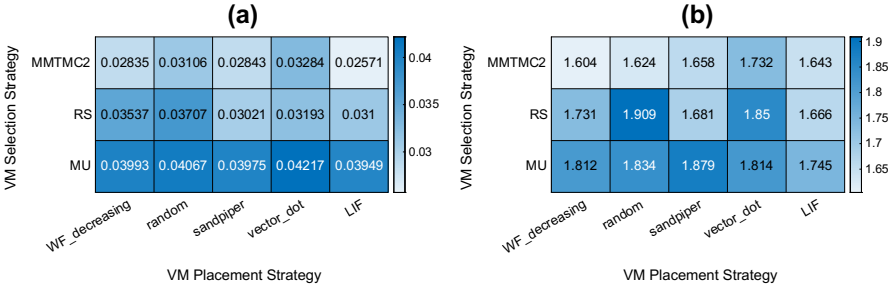



**Fig. 10** Balancer: average of the **a** $IBL^{CDC}_{avg}$ and **b** *TotalIBScore* imbalance metrics

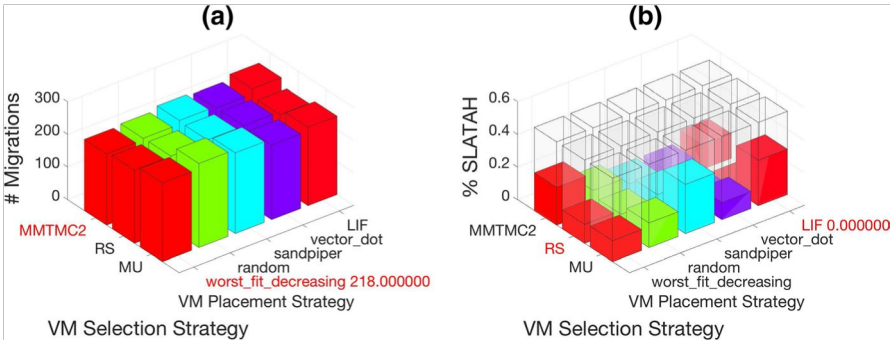


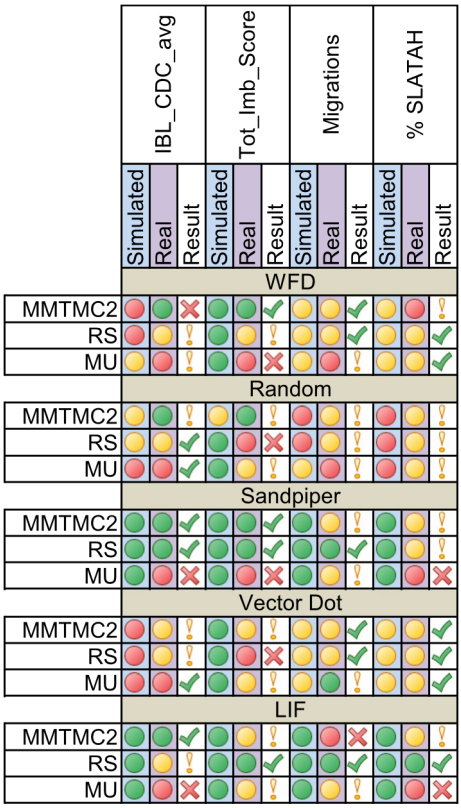

**Fig. 11** Balancer: **a** number of VM migrations and **b** SLATAH metrics

migration process. In these tests, a SLATAH percentage equal to 0.5% was obtained. In Fig. 11b, the SLATAH baseline value is showed with the transparent bars, as a complement of the rest of obtained values (colored bars). It can be seen that the LIF and RS combination generated the lowest SLATAH percentage (0%), which means that no CPU performance degradation occurred. In general, all of the evaluated strategies presented a reduction of SLATAH in more than 50% with respect to the baseline, which means the use of the VM selection-placement strategies improved the quality of service of the cloud.

### 5.3 Discussion

Figure 12 summarizes the obtained results using the CloudBench method that combines simulation and real cloud evaluation stages. It shows the combination of VMs selection (MMTMC2, RS and MU) and VMs placement (WFD, Random, Sandpiper, VectorDot and LIF) strategies, considering the different performance metrics described in Sect. 4.1.3. For each combination of VM Selection/VM Placement strategies, we show the results obtained for each performance metric ( $IBL^{CDC}_{avg}$ , *TotalIBScore*, Number of migrations and % SLATAH) in: (a) a simulated cloud, (b) a realworld cloud and (c) the result of comparing both the simulated and the real-world

**Fig. 12** Simulated and real cloud environment comparison




cloud results. On one hand, the color of the circles in columns *Simulated* and *Real* denote a classification of the obtained value, where green, yellow and red represent low, medium and high values, respectively. The three marks used in the *Result* column denote how similar were the results in both simulated and real-world environment. The red-cross mark, green-check mark and yellow-exclamation mark show if the results were the same, close or totally different, respectively. We can see that, in both simulated and real-world cloud, the RS (VM selection)-LIF (VM placement) combination produced the same results in 3 of the 4 metrics used, being $IBL^{CDC}_{avg}$ the exception. With this metric, the RS-LIF strategy presented a low imbalance value in the simulated cloud, whereas its value was medium in the real-world cloud. The combination of RS-Sandpiper also produced the same results in 3 of the 4 metrics, being % SLATAH the exception. In the simulated cloud RS-Sandpiper presented a low value of % SLATAH, whereas in the real cloud its value was medium. The combinations that included the Random VM placement strategy showed the most notable differences in the results obtained in the simulated and real cloud. For instance, in the Random-RS and

Random-MU combinations only the $IBL^{CDC}_{avg}$ metric coincided. Results in Fig. 12 confirm that RS-Sandpiper and RS-LIF are the strategies that produce very similar results in both simulated and real cloud. With the CloudBench methodology was possible to detect the strategies that have similar results in simulated and real cloud environments, which motivates its use to evaluate different simulated scenarios when there is not a real cloud available for testing.

## 6 Conclusions and future work

This article introduces CloudBench, a methodology for the evaluation and validation of VM selection and placement algorithms by integrating simulated and real-world cloud scenarios. Two tools were developed to support this methodology, the CloudBalanSim simulator and the Balancer VM manager. CloudBalanSim and Balancer allow the execution of VM selection and placement algorithms in a simulated and real-world cloud, respectively. In addition, the VM placement module of Balancer was implemented as a distributed, fault-tolerant and scalable service. The use of the Dynamically Weighted Bully (DWB) algorithm allows Balancer to dynamically choose the physical machine (PM) with more available resources as the coordinator node that will execute the VM placement and migration processes. CloudBalanSim and Balancer demonstrated its correct functionality after running the different VM selection and placement algorithms successfully. The performance metrics included in CloudBalanSim and Balancer allow us to obtain measurements of different cloud aspects such as load imbalance, SLAs violations in terms of CPU consumption in PMs and the number of VM migrations. The possibility of implementing various VMs selection and placement strategies in CloudBalanSim and Balancer allowed us to verify that some of them, such as MU and worstFitDecreasing, when are executed in scenarios with high PM consolidation, do not impact considerably the CPU performance, obtaining a degree of CPU performance degradation that does not reach 0.2%, which means a service guarantee of 99.8%. A high PMs consolidation with an acceptable CPU degradation represents a more efficient cloud resources consumption, which increases service availability and gives the possibility of saving energy by deactivating unused PMs. In scenarios with more available PMs, we found that VM placement strategies such as LIF, LIFDecreasing, Sandpiper and sandpiperDecreasing produce better imbalance levels in terms of CPU and RAM consumption, number of migrations and performance degradation, when are compared with the rest of strategies. The Random VM

placement strategy generates a large number of VM migrations, causing a significant CPU performance degradation in PMs, affecting quality of service. The CloudBench methodology proposes that the most satisfactory strategies found by CloudBalanSim are validated in a real-world cloud, in this case using the Balancer tool. In this sense, Balancer could validate the accuracy of the results obtained by CloudBalanSim, offering users more precise information. Balancer allowed us to determine that the strategy combining LIF and RS algorithms produce the best cloud load balancing with no CPU performance degradation. In addition, we found that RS-Sandpiper and RS-LIF are the strategies that produce very similar results in both simulated and real cloud, which motivates its use in simulated clouds when there is not a real cloud available for testing purposes. CloudBalanSim and Balancer offer support for running a more extensive set of tests in which the number of nodes is increased, including heterogeneous VMs (e.g., different number of cores and RAM) in the real environment. Additional to CloudBalanSim and Balancer tools, CloudBench purposes the use of an incremental learning layer that is in charge of keeping historical data (feedback). In our experiments, this layer was used during the evaluations for avoiding unnecessary tests, saving time and cloud resources.

As future work, different extensions could be made to this research. One of them is the incorporation to Balancer of other fault tolerance mechanisms such as those provided by Ceph Monitors[6] or Apache Zookeeper.[7] Another issue to be considered as future work is the extension of the CloudBench prototype implementation tools, CloudBalanSim and Balancer, to consider more physical and virtual resources, such as network and storage. With this extension it would be possible to carry out a broader and more complex analysis of the different management and load balancing strategies in private IaaS clouds, both in a simulated and in a real-world cloud. For example, the multi-resource load balancing strategies could take into account the PM's network traffic to avoid VM migrations into PMs that are receiving a lot of traffic. The integration of other metrics related to disk usage (e.g., Disk I/O) in overload detection strategies and performance metrics is also considered. Finally, our methodology could also be extended to allow the evaluation of different container load balancing strategies (container selection and placement strategies), by using Kubernetes [8] or other container management/orchestration platforms.

[6] https ://docs.ceph.com/docs/maste r/start /intro/ .

[7] https ://zooke eper.apach e.org/doc/curre nt/zooke eperO ver.html.

[8] https ://cloud .googl e.com/kuber netes /.

**Acknowledgements** This work was partially funded by the Spanish Ministry of Economy, Industry and Competitiveness under the Grant TIN2016-79637-P "Towards Unification of HPC and Big Data Paradigms" and by the Mexican Council of Science and Technology (CONACYT) through a Ph.D. Grant (No. 212677).

## References

1. Ahmed A, Sabyasachi AS (2014) Cloud computing simulators: a detailed survey and future direction. In: Advance Computing Conference (IACC), 2014 IEEE International, pp 866–872. https ://doi.org/10.1109/IAdCC .2014.67794 36
2. Beloglazov A, Buyya R (2012) Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr Comput Pract Exp 24(13):1397–1420. https ://doi.org/10.1002/cpe.1867
3. Beloglazov A, Buyya R (2015) Openstack neat: a framework for dynamic and energy-efficient consolidation of virtual machines in openstack clouds. Concurr Comput Pract Exp 27(5):1310–1333. https ://doi.org/10.1002/cpe.3314
4. Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Exp 41(1):23–50. https ://doi.org/10.1002/spe.995
5. Chen L, Shen H, Sapra K (2014) Distributed autonomous virtual resource management in datacenters using finite-Markov decision process. In: Proceedings of the ACM Symposium on Cloud Computing, SOCC '14, pp 24:1–24:13. ACM, New York, NY, USA. https ://doi.org/10.1145/26709 79.26710 03
6. Coffman EG, Garey MR, Johnson DS (1996) Approximation algorithms for bin packing: a survey. PWS Publishing Co., USA, pp 46–93
7. Durao F, Carvalho JFS, Fonseka A, Garcia VC (2014) A systematic review on cloud computing. J Supercomput 68(3):1321–1346. https ://doi.org/10.1007/s1122 7-014-1089-x
8. El Motaki S, Yahyaouy A, Gualous H, Sabor J (2019) Comparative study between exact and metaheuristic approaches for virtual machine placement process as knapsack problem. J Supercomput. https ://doi.org/10.1007/s1122 7-019-02847 -0
9. Foundation O (2016) Openstack installation guide for red hat enterprise linux and centos. http://docs.opens tack.org/mitak a/insta ll-guide -rdo/. Accessed 15 June 2016
10. Garcia-Molina H (1982) Elections in a distributed computing system. IEEE Trans Comput 31(1):48–59. https ://doi.org/10.1109/TC.1982.16758 85
11. Garg SK, Buyya R (2011) Networkcloudsim: modelling parallel applications in cloud simulations. In: 2011 Fourth IEEE International Conference on Utility and Cloud Computing (UCC), pp 105–113. https ://doi.org/10.1109/UCC.2011.24
12. Gomez-Rodriguez MA, Sosa-Sosa VJ, Gonzalez-Compean JL (2017) Assessment of private cloud infrastructure monitoring tools—a comparison of Ceilometer and Monasca. In: Proceedings of the 6th International Conference on Data Science, Technology and Applications (DATA 2017), pp 371–381. SCITEPRESS—Science and Technology Publications, Lda., Madrid, Spain
13. Gupta MK, Amgoth T (2018) Resource-aware virtual machine placement algorithm for IAAS cloud. J Supercomput 74(1):122–140. https ://doi.org/10.1007/s1122 7-017-2112-9
14. Han SH, Kim HW, Jeong YS (2019) An efficient job management of computing service using integrated idle vm resources for high-performance computing based on openstack. J Supercomput. https ://doi.org/10.1007/s1122 7-019-02769 -x
15. Hussain F, Haider SA, Alamri A, AlQarni M (2018) Fault-tolerance analyzer: a middle layer for pre-provision testing in openstack. Comput Electr Eng 66:64–79. https ://doi.org/10.1016/j. compe lecen g.2017.11.019
16. Jangiti S, Shankar Sriram VS (2018) Scalable and direct vector bin-packing heuristic based on residual resource ratios for virtual machine placement in cloud data centers. Comput Electr Eng 68:44–61. https ://doi.org/10.1016/j.compe lecen g.2018.03.029
17. Korte B, Vygen J (2006) Bin-packing. Springer, Berlin, pp 426–441. https ://doi.org/10.1007/3540-29297 -7_18

18. Kuo CF, Yeh TH, Lu YF, Chang BR (2015) Efficient allocation algorithm for virtual machines in cloud computing systems. In: Proceedings of the ASE BigData & SocialInformatics 2015, ASE BD&SI '15, pp 48:1–48:6. ACM, New York, NY, USA. https ://doi.org/10.1145/28188 69.28188 78
19. Lin W, Xu S, He L, Li J (2017) Multi-resource scheduling and power simulation for cloud computing. Inf Sci 397–398:168–186. https ://doi.org/10.1016/j.ins.2017.02.054
20. Maarouf A, Marzouk A, Haqiq A (2015) Comparative study of simulators for cloud computing. In: 2015 International Conference on Cloud Technologies and Applications (CloudTech), pp 1–8. https ://doi.org/10.1109/Cloud Tech.2015.73369 89
21. Mann ZA (2015) Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. ACM Comput Surv 48(1):11:1–11:34. https ://doi. org/10.1145/27972 11
22. Milani AS, Navimipour NJ (2016) Load balancing mechanisms and techniques in the cloud environments: systematic literature review and future trends. J Netw Comput Appl 71:86–98. https :// doi.org/10.1016/j.jnca.2016.06.003
23. Mustafa S, Nazir B, Hayat A, ur Rehman Khan A, Madani SA (2015) Resource management in cloud computing: taxonomy, prospects, and challenges. Comput Electr Eng 47:186–203. https :// doi.org/10.1016/j.compe lecen g.2015.07.021
24. Nuaimi KA, Mohamed N, Nuaimi MA, Al-Jaroodi J (2012) A survey of load balancing in cloud computing: challenges and algorithms. In: Proceedings of the 2012 Second Symposium on Network Cloud Computing and Applications, NCCA '12, pp 137–142. IEEE Computer Society, Washington, DC, USA. https ://doi.org/10.1109/NCCA.2012.29
25. Pires FL, Barán B (2015) Virtual machine placement literature review. CoRR arxiv : abs/1506.01509
26. Sato K, Samejima M, Komoda N (2013) Dynamic optimization of virtual machine placement by resource usage prediction. In: 2013 11th IEEE International Conference on Industrial Informatics (INDIN), pp 86–91. https ://doi.org/10.1109/INDIN .2013.66228 63
27. Satpathy A, Addya SK, Turuk AK, Majhi B, Sahoo G (2018) Crow search based virtual machine placement strategy in cloud data centers with live migration. Comput Electr Eng 69:334–350. https ://doi.org/10.1016/j.compe lecen g.2017.12.032
28. Singh A, Korupolu, M, Mohapatra D (2008) Server-storage virtualization: integration and load balancing in data centers. In: SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, pp 1–12. https ://doi.org/10.1109/SC.2008.52226 25
29. Thakur A, Goraya MS (2017) A taxonomic survey on load balancing in cloud. J Netw Comput Appl 98:43–57. https ://doi.org/10.1016/j.jnca.2017.08.020
30. Tian W, Xu M, Chen A, Li G, Wang X, Chen Y (2015) Open-source simulators for cloud computing: comparative study and challenging issues. Simul Model Pract Theory 58:239–254. https ://doi. org/10.1016/j.simpa t.2015.06.002
31. Tian W, Zhao Y, Xu M, Zhong Y, Sun X (2015) A toolkit for modeling and simulation of real-time virtual machine allocation in a cloud data center. IEEE Trans Autom Sci Eng 12(1):153–161. https ://doi.org/10.1109/TASE.2013.22663 38
32. Tighe M, Keller G, Bauer M, Lutfiyya H (2012) DCSIM: a data centre simulation tool for evaluating dynamic virtualized resource management. In: 2012 8th International Conference on Network and Service Management (CNSM) and 2012 Workshop on Systems Virtualiztion Management (SVM), pp 385–392
33. Wood T, Shenoy P, Venkataramani A, Yousif M (2009) Sandpiper: black-box and gray-box resource management for virtual machines. Comput Netw 53(17):2923–2938. https ://doi.org/10.1016/j. comne t.2009.04.014
34. Xu M, Li G, Yang W, Tian W (2015) FlexCloud: a flexible and extendible simulator for performance evaluation of virtual machine allocation. In: 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity), pp 649–655. https ://doi.org/10.1109/Smart City.2015.143
35. Xu M, Tian W (2012) An online load balancing scheduling algorithm for cloud data centers considering real-time multi-dimensional resource. In: 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, vol 01, pp 264–268. https ://doi.org/10.1109/CCIS.2012.66644 09
36. Xu M, Tian W, Buyya R (2016) A survey on load balancing algorithms for VM placement in cloud computing. CoRR arxiv : abs/1607.06269

37. Zhao X, Yin J, Lin P, Zhi C, Feng S, Wu H, Chen Z (2015) SimMon: a toolkit for simulating monitoring mechanism in cloud computing environments. Springer, Berlin, pp 477–481. https ://doi.org/10.1007/978-3-662-48616 -0_33
38. Zhong WTLJ (2013) LIF: a dynamic scheduling algorithm for cloud data centers considering multidimensional resources. J Inf Comput Sci 10(12):3925. https ://doi.org/10.12733 /jics2 01021 11