Which is the best algorithm for virtual machine placement optimization?

Zoltán Ádám Mann and Máté Szabó

Abstract

One of the key problems for Infrastructure-as-a-Service providers is finding the optimal allocation of virtual machines (VMs) on the physical machines available in the provider's data center. Since the allocation has significant impact on operational costs as well as on the performance of the accommodated applications, several algorithms have been proposed for the VM placement problem. So far, no objective comparison of the proposed algorithms has been provided; therefore, it is not known which one works best or what factors influence the performance of the algorithms. In this paper, we present an environment and methodology for such comparisons and compare seven different algorithms using the proposed environment and methodology. Our results showcase differences of up to 66% between the effectiveness of different algorithms on the same real-world workload traces, thus underlining the importance of objectively comparing the performance of competing algorithms.

This paper was published in *Concurrency and Computation: Practice and Experience*, volume 29, issue 10, 2017.

1 Introduction

Data center operators rely increasingly on virtualization technology to enable the safe co-existence of multiple applications or application components on the same physical machine (PM) in the form of virtual machines (VMs). The ability to allocate several VMs on the same PM makes it possible to achieve a healthy utilization of the available physical resources, thus amortizing the capital investments of purchasing them [7, 35].

The load of a VM is typically not constant; especially the CPU load is known to exhibit large variance over time [67, 73]. Therefore, the optimal allocation of VMs to PMs also changes dynamically. The importance of dynamic VM placement is intensified by the high power consumption of data centers (DCs). According to a recent study, DC electricity consumption in the USA alone will increase to 140 billion kWh per year by 2020, costing US businesses 13 billion USD annually in electricity bills and causing the emission of nearly 100 million tons of CO_2 per year [58]. Power consumption can be significantly reduced by consolidating the workload to the minimum number of necessary PMs and switching unneeded PMs into a low-power mode (e.g., sleep or hibernate). Using live migration technology, VMs can be moved between PMs without noticeable downtime [77]. Therefore, DC operators regularly re-optimize the mapping of VMs to PMs, and perform the necessary migrations to get to the newly determined placement. This way, the provider can adapt resource usage to the workload's resource needs: in times of low demand, the workload will be consolidated to a low number of PMs, thereby saving a considerable amount of energy; in times of high demand, the VMs will be spread across many more PMs so that their resource requirements – and ultimately, service level objectives – are satisfied [78].

The algorithm used by the operator for re-optimizing the placement of VMs has large impact on multiple vital metrics [64]:

- **Energy consumption.** A good VM placement optimization algorithm achieves low overall DC energy consumption, mainly by consolidating the VMs to as few PMs as possible (without violating performance objectives – see below). Beside the number of active PMs, there are also some other levers for saving energy. In a typical DC, the PMs are not homogeneous; e.g., there can be older and newer machines or machines of different type. As a result, PMs may have different power efficiency, i.e., the energy consumption per instruction can be different. A good VM placement optimization algorithm can take advantage of this by favoring PMs with better power efficiency. Moreover, a PM's power consumption is not constant but depends on the PM's load [69]. This fact, together with the heterogeneity of PMs makes matters more complicated: different PMs may have different load–power characteristics, and hence the question whether a given VM leads to more energy consumption on PM A or PM B may also depend on the two PMs' load and thus indirectly also on the placement of the other VMs. A good VM placement optimization algorithm should take this into account and strive to reach the placement with overall minimal energy consumption [3].

- **Application performance.** Too aggressive consolidation may lead to congestion or overload of a PM's resources. In that case, the accommodated VMs cannot obtain the amount of resources they would need, resulting in performance degradation of the applications running in those VMs, which in turn likely leads to violation of Service Level Objectives (SLOs). Depending on the contractual terms between the provider and its customers, SLO violation may also lead to a financial penalty for the provider, but in any case, it adversely impacts customer satisfaction. Therefore, a good VM placement optimization algorithm minimizes the frequency and duration of resource overload events [81].

  It should be noted that, being on the level of Infrastructure-as-a-Service, application-level SLOs are not known to the VM placement algorithms. Rather, they assume that, as long as VMs obtain the resources they request, SLOs will be satisfied.

- **Migration overhead.** Although live migration minimizes the period in which the VM is unresponsive to the sub-second range, the migration takes actually significantly longer and incurs non-negligible overhead during this time in terms of both network traffic and additional PM load [25,26,77,84], also resulting in extra energy consumption [48]. Therefore, a good VM placement optimization algorithm has to minimize the number of migrations, i.e., the number of VMs whose placement must be changed.

As can be seen from the above, VM placement is an important optimization problem with far-reaching impact. Unfortunately, it is also a computationally very challenging problem: a sufficiently general formulation of the VM placement problem includes the well-known bin-packing problem as a special case, and hence it is strongly NP-hard[1], so that the existence of an efficient exact algorithm is very unlikely [50]. On the other hand, VM placement algorithms must be able to solve problem instances with thousands of PMs and VMs in reasonable time (in seconds, or at most a few minutes) in order to be practical. As a consequence, most of the algorithms that have been proposed so far are heuristics.

Since there is no theoretical guarantee on the effectiveness of most of the proposed heuristics, empirical assessment and comparison of the algorithms should play an important role. Unfortunately, very few of the published works include a thorough empirical evaluation of their contribution, let alone an unbiased comparison of several competing algorithms. The empirical evaluation in many papers is very limited, e.g.

- completely lacks comparison to previously published algorithms;
- comparison is made only with trivially non-competitive algorithms (e.g., random or round-robin placement, or greedy algorithms that do not take into account some vital aspect of the problem);

[1]Strongly NP-hard means that the problem remains NP-hard even if the numbers appearing in it are constrained between polynomial bounds. Under the P≠NP assumption, this precludes even the existence of a pseudo-polynomial algorithm – i.e., an algorithm the runtime of which is polynomial if restricted to problem instances with polynomially bounded numbers.

- only different versions of the authors' algorithm are compared against each other;
- comparison is done only on very small or very special problem instances.

It has to be noted though that in many cases it is not at all easy to make a meaningful comparison with competing algorithms. The main obstacles are:

- The VM placement problem has many different flavors with subtle but important differences between them [51]. Thus, different algorithms may actually solve slightly different problems. It is not always obvious which algorithms solve exactly the same problem and can thus be used for comparison.
- Only few of the proposed algorithms have either publicly available source code or are described in the respective papers in sufficient detail to allow reproduction.
- It is generally not feasible to test VM placement algorithms on large-scale real-world testbeds: most authors do not have access to real DCs and even if they have, applying a placement algorithm that is still in an experimental state to a real DC would be dangerous.
- In lieu of real-world test environments, workload traces of such could be used. However, there are very few publicly available traces. The few that are available are quite different from each other both in format and content, making it tedious to experiment with multiple traces.
- There is no set of widely accepted benchmarks nor benchmarking methodology for the VM placement problem like in some of the more mature fields of computing.

As a result, literally hundreds of algorithms have already been proposed for (different versions of) the VM placement problem, but currently we have no way to tell how they compare to each other in terms of solution quality.

In this paper, we make a first step towards remedying this situation. Our contributions are as follows:

- We present a **test environment** for assessing the effectiveness of VM placement algorithms. This test environment builds on an existing cloud simulator, but extends it with several further components that are needed for reproducible experiments, like converters for publicly available workload traces and a workload generator.
- Using this environment, we undertake a detailed empirical **comparison of seven algorithms** that address the same version of the problem. Beyond the results of the comparison itself, it also exemplifies the methodology that we propose for this purpose, thus it can be seen as a **methodological template**.

Our measurement results reveal not only how the assessed algorithms compare to each other, but also the factors that have the largest influence on their performance. In some cases, we found differences of roughly 66% between the effectiveness of competing algorithms on the same real-world workload trace, which again underlines the importance of such comparisons.

In the remainder of the paper, we first review related work in Section 2. Section 3 presents our test environment, followed by the presentation of our measurements in Section 4, while Section 5 concludes the paper.

2 Related work

In the last couple of years, there has been tremendous interest in resource optimization for cloud computing systems. As described in our recent survey [49], the problem formulations used in these works differ in several aspects. We mention here the most important ones:

- **Resource types.** Most works focus on the CPU as the most critical resource and characterize PMs in terms of their CPU capacity and VMs in terms of their CPU load [4–7,9,10,13,33,36,40,44,82,83]. On the other hand, some works make the problem multi-dimensional by also considering some other resource types like memory and I/O [8,12,28,31,43,56,57,81,92] or by differentiating between the CPU cores [53].

- **Considered VM set.** Most works consider the placement of all VMs in the DC at once [5, 8, 23, 33, 35, 40, 57, 68, 71, 74, 88]. However, some also consider the placement of a single VM [76,81] or a set of VMs belonging to the same application [13,39,79].

- **New placement vs. re-optimization.** For the given VM set (whether it is the set of all VMs of the DC, a single VM, or the VMs of an application), there are two different problem variants: either the initial placement of the VMs needs to be determined [1,4,8,39,68,81], or their existing placement is to be re-optimized to adapt it to the changed resource requirements of the workload [10,23,33,35,40,71,74,83,88]. The two problems are similar; the main difference is that re-optimization has to take into account the cost of migrating a VM from its old host to a new one, whereas this is no concern in the case of initial placement.

- **Objective(s).** Minimizing energy consumption is a central objective in most works. However, there are differences in the level of detail that energy consumption is modeled with. Several works consider the number of active PMs as an indication of energy consumption [10, 13, 31, 68, 83, 92]. Some also take into account the load-dependent dynamic power consumption of PMs [6, 22, 28, 33, 40, 42, 52, 78, 82]. Beyond energy consumption, a further objective in some works is to minimize the number of overloaded PMs because of the performance degradation that results from overloads [6, 10, 13, 81, 83, 86]. Some works also considered the cost of migration of VMs [6,13,31,65,78,82], or reliability [72].

- **SLA handling.** Many works assume that the SLO is to provide to each VM the amount of resources that it requires; therefore, they pack VMs into PMs so that the total resource demand of the VMs is not more than the capacity of the PM [24, 59, 75, 86–88]. Some relax this and allow overload of the PM resources but try to minimize the time when this happens [5, 6, 83] or the probability of such overload [14, 21, 27, 30]. There were also some attempts to handle SLOs beyond resource overload, like availability [19], execution time [9,69], or response time [33,40], and more generally, the connection between high-level user requirements and system-level SLOs [61].

In terms of the proposed algorithmic techniques, some works suggested exact methods but the majority applied heuristics. The proposed exact methods rely almost always on some form of mathematic programming (e.g., integer linear programming) and appropriate solvers [4,33,34,68,90]. Unfortunately, these approaches do not scale to practical problem sizes.

Many different heuristics have been proposed from simple greedy algorithms to evolutionary methods. As already mentioned, the VM placement optimization problem is closely related to bin-packing, for which some simple packing heuristics like First-Fit (FF), Best-Fit (BF), and First-Fit-Decreasing (FFD) are known to deliver good results. Accordingly, several researchers have suggested to adapt such packing heuristics to the – more complex – VM placement optimization problem [5, 6, 10, 33, 40, 47, 66, 81–83]. Metaheuristics have also been suggested, e.g., simulated annealing [38,54], genetic algorithms [32], particle swarm optimization [45], ant colony optimization [28], and biogeography-based optimization [46,91]. Some authors proposed proprietary heuristics. Some of them are simple greedy algorithms [71,86] or straight-forward selection policies [4,5,74], others are rather complex [40,57].

There have already been some attempts to compare the effectiveness of multiple heuristics empirically; however, these were very limited in the coverage of algorithms, the used test data, or simply had different focus from our work. Shi et al. compared three different versions of FFD-type packing and three versions of another greedy algorithm, using a randomly generated workload [74]. Chowdhury et al. compared five different packing heuristics, using a real-world workload trace [22]. Villegas et al. compared eight simple policies for VM allocation, but from a customer perspective [85].

The efficacy of VM allocation algorithms is usually evaluated with the help of cloud simulators [80]. The most widely used simulator is CloudSim [18], which models most of the aspects of a cloud system that are relevant for VM allocation. Beside being used as is, CloudSim also serves as the basis for some specialized simulation environments, like CloudReports [70], ContainerCloudSim [63], or a simulation of scaling policies [2]. A similarly well established simulator is SimGrid [20]: as a simulation framework for distributed systems, it has existed for many years, but it has only recently been extended with the means necessary to model VM placement [37]. DISSECT-CF is another alternative that has appeared recently and promises improved simulation speed [41], but is still in an earlier phase of its development. There are also simulators with special emphasis on the networking part of cloud computing, like GreenCloud [11].

3 Test environment

It is important to carefully arrange a well-defined environment for carrying out the empirical assessment of algorithms, so that the experiments are reproducible and the results are conclusive. By 'environment,' we mean not only the physical environment in which the tests are carried out, but also all the tools, data, and settings that are necessary for the empirical assessment.

Unfortunately, it is not feasible (at least not with reasonable costs) to test the effectiveness of VM allocation optimization algorithms in a large-scale real-world environment. Therefore, a cloud simulator is used as the basis for our test environment. For this purpose, we chose CloudSim [18], the most widely used cloud simulator. CloudSim is a mature and established simulation framework that simulates all entities that we need – like PMs and VMs – mostly in sufficient detail, and is open-source, so that it can be extended as necessary. As a bonus, some algorithms already have an available CloudSim implementation that can be used directly in our experiments.

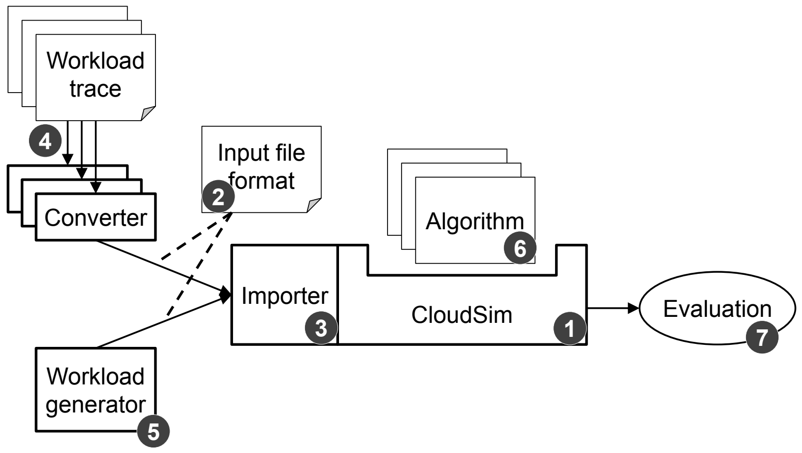


Figure 1: Overview of the proposed test environment. It uses CloudSim as central simulation engine, and extends it with facilities to import data files of special format, to convert existing workload traces to the given format, and generate test data in the given format. Different VM placement algorithms can be compared on the input data by means of defined evaluation metrics.

Figure 1 shows an overview of the test environment. The individual components are described in the next subsections according to the order of the numbering in the figure.

3.1 CloudSim

CloudSim was developed by researchers at the Cloud Computing and Distributed Systems Laboratory of the University of Melbourne [18] on the basis of their previous grid simulator called

GridSim [16]. CloudSim is written in Java and its source code and additional resources are publicly available under the LGPL license from the project homepage[2]. CloudSim is probably the most widely used cloud simulator.

CloudSim comes with a built-in VM placement algorithm, implemented in the method PowerVmAllocationPolicyMigrationAbstract.optimizeAllocation and the methods that it calls for host overload detection and similar subtasks. This algorithm was developed by Beloglazov et al. [5,6]. In order to change CloudSim's VM placement behavior, this method (or some of the other methods that it calls) needs to be overridden.

Also some other VM placement algorithms that have been proposed in the literature have a publicly available CloudSim implementation. But unfortunately, they are not necessarily implemented using the same CloudSim version: e.g., the algorithm of Lago et al. is implemented in CloudSim 2.0 [44], whereas the algorithm of Beloglazov et al. is implemented in version 3.0 [5]. For our experiments, we used CloudSim 3.0.3, and ported algorithm implementations from earlier versions where required.

CloudSim also has some limitations. Most notably, networking aspects are modeled only rudimentarily: bandwidth is considered to be a characteristic of PMs, and there is no way to model the bandwidth of individual network elements like switches or links. For this reason, there are extensions to CloudSim that add networking support [29]. For our current purpose, these limitations are not problematic, but if algorithms for more sophisticated problem variants are to be compared, this may require substantial modifications to CloudSim or even replacing it with another simulator that is more appropriate for the given purpose.

3.2 Input file format

Another limitation of CloudSim, which is more problematic with respect to our purposes, is how it handles input data of the simulation: some parameters have to be hard-coded in the program, for some others, a simplistic file interface is provided. In order to conduct a large number of simulations with different input parameters in a reproducible manner, a structured way for specifying the inputs outside the code is indispensable.

For the sake of simplicity and easy portability, we decided to use a set of files for storing all input data, including PM and VM data, just like simulation parameters. Because of the central role of these files, special care must be taken when designing the file formats. In particular, we considered the following requirements towards the file format:

- **Easy experimentation with parameters.** The most important requirement is that the file format should make it easy to run simulations with changed settings, e.g., same set of VMs with different set of PMs, or same set of VMs and PMs but with different workload patterns.
- **Reproducibility of experiments.** The input files should contain every detail that is necessary to reproduce an experiment.
- **Human readability.** Although for the simulations themselves, the files must be only machine-processable, but it is often useful if the experimenter can check or change some parameters directly in the files, without needing some special tools.
- **File sizes.** Workload traces can easily become very large (many gigabytes), which can make storing and manipulating them rather cumbersome. Therefore, one must be careful not to add much overhead in terms of file size.
- **Extensibility.** As mentioned before, the VM placement problem has many different versions, which need partly different input data (e.g., some require a communication matrix describing the intensity of data transfer between pairs of VMs, whereas others do not). Also, the available workload traces have somewhat different scope (see Section 3.4), e.g.,

[2] http://www.cloudbus.org/cloudsim/

some include only CPU load values, others also include load values for other resources like memory or disk. Therefore, the file formats must be chosen in such a way that – beyond a small set of mandatory data – further data may be present optionally. It should also be easy to add further fields in the future, without invalidating past experiments and their input files.

Unfortunately, existing formats like the Standard Workload Format (SWF)[3] or the Grid Workload Format (GWF)[4] were created for somewhat different purposes, and hence do not fulfill our requirements. For example, SWF and GWF files do not contain dynamic VM information (temporal evolution of resource load).

After having analyzed the above requirements, the data available in the form of workload traces (see also Section 3.4), and the data requirements of the algorithms to be tested (see Section 3.6), we came to the conclusion that different types of input should be in separate files and need different formats. Specifically, we differentiate among the following types of input data:

- **PM characteristics.** In a typical DC, there are many PMs of the same type, that have the same characteristics. Therefore, our input file consists of the description of PM types. For each PM type, the list of PMs of the given type is given (individual PMs are identified by a unique ID). Moreover, the characteristics of the PM type are given; currently, these are
  - Number of cores
  - CPU capacity per core, in MIPS (million instructions per second)
  - RAM capacity, in MB
  - Disk capacity, in MB

  In order to ensure both machine-processability and human-readability, we use the YAML format[5] to store these data. An example can be seen in Figure 2. The use of YAML also gives us the necessary flexibility: if some elements are missing or if some elements are added later, the files remain syntactically correct.

```
- Host:
    ID: [0,1,6,7,4]
    Cores: 2
    CpuCapacity: [1860,1860]
    RAM: 4096
    DiskCapacity: 1000000
- Host:
    ID: [2,3,5]
    Cores: 1
    CpuCapacity: [2600]
    RAM: 4096
    DiskCapacity: 1000000
```

Figure 2: Excerpt from a YAML file describing PM data. Each element specifies a PM type with its capabilities (number of CPU cores, capacity of the CPU cores, memory size, disk capacity) and gives the list of available PMs of the given type.

- **VMs' static characteristics.** Similarly as for PMs, a file describes the static attributes of the VMs in YAML format. Again, it is possible to have several instances of the same type, so the file actually specifies VM types, and gives the list of IDs of specific VMs belonging to each type. The attributes stored for each VM type are:

[3] http://www.cs.huji.ac.il/labs/parallel/workload/swf.html
[4] http://gwa.ewi.tudelft.nl/grid-workload-format/
[5] http://www.yaml.org/spec/1.2/spec.html

- Start time of the VM, from the starting point of the simulation, in milliseconds
- End time of the VM, from the starting point of the simulation, in milliseconds
- Number of cores
- Requested CPU capacity per core, in MIPS
- Requested RAM capacity, in MB
- Requested disk capacity, in MB

An example is given in Figure 3. Note that this file stores only the requested capacities, as specified by the user at the time of requesting the VM. The actually used capacities are stored in the dynamic part, see below.

```
- VM:
    ID: [0,1,2,3]
    StartTime: 0
    EndTime: 100000
    Cores: 1
    CpuCapacity: [2000]
    RAM: 1740
    DiskCapacity: 2500
- VM:
    ID: [4,5]
    StartTime: 0
    EndTime: 100000
    Cores: 2
    CpuCapacity: [500,600]
    RAM: 1740
    DiskCapacity: 2500
```

Figure 3: Excerpt from a YAML file describing static VM data. Each element specifies a VM type with its resource requirements (requested number of vCPU cores, requested capacity of the CPU cores, requested memory size, requested disk capacity) and timing attributes (start and end time), and gives the list of VMs of the given type.

- **VMs' dynamic characteristics.** This part consists of the time-dependent resource usage values of the VMs. In most cloud systems, such values are sampled at regular time intervals, e.g., every five minutes. If the number of VMs is high and the trace is collected over a long period of time, the resulting amount of data can be voluminous; therefore, we chose to use a low-overhead binary format in this case. The data are stored in *blocks*, where each block consists of a header and a body. The header contains the following pieces of information:
  - ID of the VM
  - Sample period in milliseconds
  - Number of samples in the block

  The body of the block contains for each sample one byte that stores the usage value at the given point of time, as percentage of the requested capacity. An example is shown in Figure 4. The same format can be used for any resource type, with the restriction that a single file contains only data for one resource type (e.g., one file for CPU and another one for memory). As a convention, the resource type is given in the filename; more importantly, the mapping of resource types to files is specified in the configuration file (see below under "Parameters of the test").

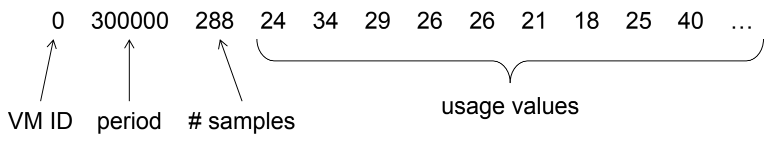


Figure 4: Example for a block of dynamic VM data, consisting of the ID of the VM, the sample period, the number of samples in the block, and the list of actual usage values (samples)

- **Optionally: PM and VM topologies.** For some versions of the VM placement problem, the topology of the PMs (and potentially also other elements of the DC network, e.g., switches) and/or the communication relationships among VMs may be of interest. For this purpose, appropriate formats already exist, so that there is no need to define a new format. Specifically, the BRITE format[6] that CloudSim already uses for specifying network topologies is adequate for our purposes.
- **Parameters of the test.** One more file is used which contains the parameters that drive the given test case. This configuration file must be easily readable and editable, so once again we settled for using YAML. In our current implementation, the configuration file only contains the names of the files that contain the PM data, the static VM data, and the dynamic VM data for each considered resource type; see Figure 5 for an example. If necessary, further parameters can also be configured here, e.g., the algorithms to run and their parameters.

```
hosts: testdata/all/1200_pm_hosts.txt
vms: testdata/all/control_planetlab_vms.txt
workload_cpu: testdata/all/control_planetlab_cpuusage.txt
```

Figure 5: Example configuration file, specifying the location of the PM data, static VM data, and dynamic VM data files

3.3 Importer

After having defined the formats of the input files, the next step is to enable CloudSim to import files of these formats. For this purpose, we had to extend it with a new Importer module. In order to read and process YAML files, we used the SnakeYAML Java library[7]. For the files describing dynamic VM data, we had to implement our own reader, but this was easy as the file format is very simple.

3.4 Workload traces and converters

In order to obtain practically relevant results, it is important to use realistic test data. This is especially true for the dynamic VM data, because real VM resource consumption is known to be oscillating in highly non-trivial ways [67]. Understanding how different VM placement algorithms react to typical workload change patterns is therefore of key importance.

Unfortunately, few relevant workload traces are publicly available, due to privacy concerns and business competition. Although some grid workload traces were made publicly available in the past (e.g., in the Parallel Workloads Archive, the Grid Workloads Archive, or the Grid Observatory), but these are unfortunately mostly not usable for our purposes because they do not stem from a virtualized environment and/or lack data that would be vital for the purposes of VM allocation.

We are aware of only three sets of workload traces that are publicly available and relevant for VM placement algorithms. In the following, we describe these and their integration into our test environment. A comparison of the three datasets is shown in Table 1.

[6] http://www.cs.bu.edu/brite/user_manual/node29.html
[7] http://bitbucket.org/asomov/snakeyaml

Table 1: Summary of the used real-world workload traces

Origin	Virtualized	Size	Duration	PM data	Static VM data	Dynamic VM data		
						CPU	Memory	Disk
PlanetLab	yes	1000 VMs	10 days	no	no	yes	no	no
Google	no	12000 PMs	29 days	yes	yes	yes	yes	yes
Bitbrains	yes	1750 VMs	4 months	no	no	yes	yes	no

3.4.1 PlanetLab – CoMon

PlanetLab is a global initiative for fostering research in distributed computing. It consists of more than 1300 physical nodes worldwide that run several co-located but isolated user tasks (so-called slices) using virtualization [62].

CoMon was a monitoring system developed to gather and process data from PlanetLab nodes [60]. The server of the CoMon project collected a lot of useful information – including the temporal change of resource needs – from about 600 nodes for several years. Unfortunately, the server broke down, also effecting the end of the CoMon project. As far as we know, the archive of the CoMon project is not available anywhere anymore, except for the CPU usage data from 10 days and about 1000 VMs, which were used by Beloglazov et al. for their experiments [6] and were conserved in the folder examples/workload/planetlab of CloudSim.

Beloglazov et al. were focusing on CPU load, and hence conserved only these data. These are stored in simple text files: one file per VM, in which each line contains a single number, the CPU load in the given sample, as a percentage of the requested capacity. It was straight-forward to create a converter that transfers these data into our format for dynamic VM data. Note however that PM data or static VM data are not available in this trace, so that these must be replaced by artificially generated data (just as Beloglazov et al. did in their experiments).

3.4.2 Google cluster data

In 2011, Google made a dataset publicly available, which contains resource and workload data from a cluster of roughly 12,000 PMs in a period of 29 days [67]. For data protection reasons, the published dataset contains only relative numbers and all strings are obfuscated. Fortunately, this does not hinder the usage of the dataset for our purposes.

In contrast to the other available traces, the Google dataset also contains some information about the PMs. For each PM, its – normalized – CPU and memory capacity are given.

The workload is comprised of jobs; each job consists of one or more tasks. Tasks are independently running software units and also the dynamic data are given for the tasks; therefore, the notion of tasks may correspond to VMs in our terminology. However, there is no evidence that the tasks in the Google dataset are actually VMs. In fact, given that the majority of tasks have tiny resource consumption in the trace, it is likely that they are not VMs on their own. Nevertheless, the temporal development of the tasks' resource needs may give important insight that can also be useful in the context of VM placement. But in order to have useful input for VM placement, too small tasks may have to be filtered out or aggregated to some bigger units.

For each task, the requested CPU, memory, and disk capacity are given. More importantly, the actual CPU, memory, and disk size of the tasks was regularly measured with some sample period. (In most cases, the sample period is 5 minutes, but there are some deviations.) Within the sample period, resource load was measured every second, but the trace contains only the maximum and average values for each sample period, plus, for the CPU load, also one randomly selected measurement result for each sample period.

The conversion from Google's data into the format described in Section 3.2 involves, beside the obvious syntactic changes, also some further transformation:

- Converting the relative numbers in the trace into absolute numbers by assuming some given maximum values.

- Removing tasks the resource usage of which is below a given threshold.
- Filtering dynamic VM data to focus on a shorter time frame (e.g., one day), with the aim of reducing data size.

3.4.3 Bitbrains

A Dutch service provider specialized in managed hosting for enterprise customers, Bitbrains released a dataset containing workload data for altogether 1750 VMs from its hosting center, representing business-critical enterprise applications [73]. In contrast to the two datasets described previously, the Bitbrains trace contains absolute resource consumption values.

The trace is available from the Grid Workloads Archive (but unlike the other traces in the Archive, it is in a proprietary format, not in the Grid Workload Format). It is comprised of one file per VM, describing mainly the VM's dynamic data, sampled every 5 minutes. These data include the used CPU and memory of the VM at the given point of time, as well as disk and network I/O throughput values.

PM data are completely missing. Static VM data are not provided explicitly, but the starting time and the number of CPU cores can be extracted from the available data. However, the requested capacities are not available. Accordingly, our converter mostly creates dynamic VM data from the dataset, and fills only a small part of static VM data (VM IDs, starting times).

3.5 Workload generator

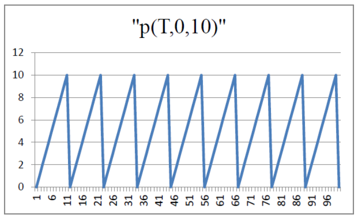
Given the sparse availability of real-world test data, a useful option is to generate test data by means of an appropriate generator. This has two major advantages: (i) we can generate as much test data as needed and (ii) by appropriate parametrization of the generator, we can create test data tailored for specific experiments.

More specifically, we are targeting two distinct aims, leading to two distinct testing scenarios with differing test data requirements. In the first scenario, we would like to test the VM placement algorithms on realistic workloads. For this purpose, we need to be able to generate workload traces that are similar to the real-world ones. In the second scenario, we would like to test how the algorithms react to situations that rarely occur in practice, hence may not appear in the specific real-world traces that are readily available, but might still happen so that it is important to understand how the different algorithms behave in such cases. For this purpose, we need to be able to generate workloads with given patterns, e.g., periodic load changes with configurable frequency, slowly or quickly increasing/decreasing load etc. In accordance with these two scenarios, we implemented two different mechanisms: a realistic workload generator and an artificial pattern generator.

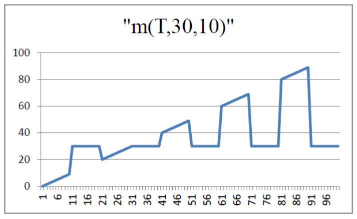
3.5.1 Realistic workload generator

Finding out what characterizes realistic workload traces is an interesting problem on its own. Yin et al. identify burstiness and self-similarity as the key characteristics of realistic workloads [89]. Minh et al. found five such characteristics: long-range dependence, periodicity, temporal burstiness, bag-of-tasks behavior, and correlation between runtime and parallelism [55].

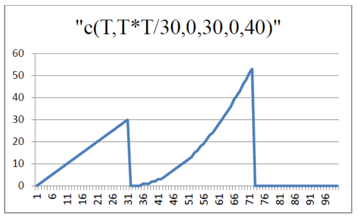
In order to generate workloads with such characteristics, a good possibility is to use Markov modulated Poisson processes [15,89]. In this approach, requests are generated by Poisson processes with different intensity. Each intensity corresponds to a state of a Markov chain, with given transition probabilities. In the simplest case, the Markov chain consists of only two states: a low-intensity and a high-intensity state. This is useful to model a normal load level versus sudden peaks in demand (the flash crowd effect).



(a) Periodic repetition of the function $T \mapsto T$ in the interval [0,10]



(b) Periodic merging of the functions $T \mapsto T$ and $T \mapsto 30$ with period 10



(c) Concatenation of $T \mapsto T$ in the interval [0,30] and $T \mapsto T^2/30$ in [0,40]

Figure 6: Example patterns that can be generated with the proposed artificial pattern generator

```
Expr =   T | <random> | <number>
       | ( Expr )
       | Expr Op Expr
       | TrigFun ( Expr )
       | p ( Expr , <number> , <number> )
       | m ( Expr , Expr , <number> )
       | c ( Expr , Expr , <number> , <number> , <number> , <number> ).

Op = + | - | * | / | ^.
```

Figure 7: Syntax for specifying workload patterns to be generated by the proposed artificial pattern generator. The meaning of the “p”, “m”, and “c” operators is exemplified in Figure 6

3.5.2 Artificial pattern generator

In order to test the effect of specific workload patterns, we also implemented a customizable pattern generator. It can generate functions of the variable T (time), consisting of the following building blocks:

- Basic arithmetic operations
- Random numbers sampled uniformly from [0,1]
- Trigonometric functions
- Periodic repetition of a function on a given interval (see Figure 6a)
- Periodic merging of two functions on given intervals (see Figure 6b)
- Concatenation of two functions on given intervals (see Figure 6c)

In order to facilitate experimentation, we defined a simple language for describing workload patterns and implemented a parser to interpret such descriptions. The syntax of this language is defined using Backus-Naur form in Figure 7.

3.6 Algorithms

The VM placement algorithms to be tested must be implemented in CloudSim, overriding its default placement algorithm implemented in the method PowerVmAllocationPolicyMigrationAbstract.optimizeAllocation and the methods that it calls. The algorithms must take as input the current placement of VMs and return a new, optimized placement. There is no other restriction whatsoever on the algorithms that can be tested.

However, in our experiments, we decided to focus on one family of VM placement algorithms that can be meaningfully compared to each other. First we had to make sure that each of the tested algorithms solves the same version of the VM placement problem. As discussed in Section 2, this is not so easy since many different versions of the problem have been considered. Therefore, we settled for a basic version of the problem that can be seen as the "lowest common denominator" of the versions typically addressed by VM placement algorithms. Using the taxonomy of [51], we chose the *Single-DC* | *1D(CPU)* | *Reopt(full)* | *Min(TotStatDynPow)* variant, meaning that all PMs are in a single DC, the CPU is considered as the only resource type, the placement of all VMs of the DC must be reoptimized, and the primary objective is to reduce overall energy consumption.

As explained in Section 2, also many different types of algorithms have been suggested for VM placement. The most popular has been the family of *packing heuristics* inspired by algorithms for bin packing or other related problems (where some of these heuristics are also proven to deliver near-optimal results [50]). Since the running time of these algorithms is in the same order of magnitude, it makes sense to compare the quality of the results they deliver.

Algorithm 1 General packing algorithm for VM placement

```
1: $L \leftarrow$ {VMs that need a (new) hosting PM}
2: while $L \neq \emptyset$ do
3:     $v \leftarrow L$.PopNextVm()
4:     $Q \leftarrow$ {PMs}
5:     $found \leftarrow false$
6:     while $Q \neq \emptyset$ and not $found$ do
7:         $p \leftarrow Q$.PopNextPm()
8:         if $p$ has enough capacity to host $v$ then
9:             place $v$ on $p$
10:            $found \leftarrow true$
11:        end if
12:    end while
13: end while
```

The general flow of the investigated algorithms is depicted in Algorithm 1. The algorithms differ mainly in the order in which the VMs are considered (the PopNextVm() method in Line 3 of the pseudo-code) and the order in which the PMs are considered for a given VM (the PopNextPm() method in Line 7 of the pseudo-code). The resulting overview of the considered algorithms can be seen in Table 2. More details are provided in the following paragraphs.

3.6.1 Beloglazov

CloudSim's default VM placement behavior is Beloglazov's algorithm [6]. Beside the placement algorithm (called Modified Best Fit Decreasing, MBFD) itself, Beloglazov et al. also experimented with different ways of determining the set of VMs to migrate (see Line 1 in Algorithm 1). The basic idea is to remove all VMs from underloaded PMs so that they can be switched off and remove some of the VMs from the overloaded PMs so that they will not be overloaded. This requires determining when to consider a PM to be overloaded and which VMs to remove from an overloaded PM. Beloglazov et al. evaluated several possible approaches. They found that the best results are achieved if overload detection is performed using local regression (see Section 4.2.6 for more details) and for overload mitigation, the VMs with smallest estimated migration time are selected until the PM is not overloaded anymore.

3.6.2 Lago

The Lago Allocator specifies a sophisticated set of rules for selecting the next PM for the current VM:

Table 2: Overview of the investigated algorithms (*Name* is based on the name of the first author)

Name	Paper	Selecting the next VM	Selecting the next PM
Beloglazov	[6]	Highest CPU load	Smallest increase in energy consumption
Lago	[44]	Highest CPU load	Highest energy efficiency (plus further rules for tie-breaking)
Guazzone	[33]	Highest CPU load	Highest free CPU capacity (plus further rules)
Chowdhury	[22]	Highest CPU load	Highest increase in energy consumption
Shi – PU	[74]	Highest CPU load on the PM with lowest utilization ratio	Highest utilization ratio
Shi – AC	[74]	Highest CPU load on the PM with lowest absolute capacity	Highest absolute capacity
Calcavecchia	[17]	VMs of the most loaded PM	All other PMs

- The principal criterion is to choose the PM with the highest energy efficiency, where energy efficiency is the ratio of CPU capacity to peak power consumption.
- In case of a tie, the PM with the lowest energy consumption is selected.
- If there is still a tie, the PM with highest CPU utilization is chosen.
- If there is still a tie, the PM with highest CPU capacity is chosen.

For selecting the next VM, the Lago allocator does not specify any rule, hence the default behavior of CloudSim applies, which means that the VMs are considered in decreasing order of CPU load.

3.6.3 Guazzone

In the first place, Guazzone et al. approached the VM placement problem with mixed-integer nonlinear programming methods, but then resorted to a best-fit-decreasing heuristic because of scalability issues [33]. PMs are sorted according to three criteria:

- Powered-on PMs precede powered-off PMs.
- Within the two groups based on power state, PMs are sorted in decreasing order of free CPU capacity.
- In case of a tie, PMs are selected in increasing order of idle power consumption.

3.6.4 Chowdhury

Chowdhury et al. implemented and compared multiple VM placement algorithms, although the comparison was very limited, using only a single workload and DC configuration [22]. From those algorithms, we take one that performed well in their comparison (there was no clear winner), named Modified Worst Fit Decreasing VM Placement (MWFDVP). Regarding PM selection, it does the opposite from the algorithm of Beloglazov et al.: it prefers the PM with the highest increase in energy consumption. While this may seem counter-intuitive, the authors argue that also in case of bin-packing the worst-fit heuristic has its merits over the more intuitive best-fit heuristic because worst-fit decisions may lead to better situations in the future.

3.6.5 Shi

Similarly to Chowdhury et al., also Shi et al. implemented and evaluated multiple packing algorithms [74]. Also here, the comparison was rather limited and there was no clear winner. We selected two algorithms that performed quite well, the PercentageUtil (Shi − PU) and the AbsoluteCapacity (Shi − AC) algorithms. In both cases, the PMs are sorted according to the given metric (utilization and capacity, respectively), and then the algorithm iteratively attempts to free the smallest PM by trying to migrate its VMs to the biggest PMs.

3.6.6 Calcavecchia

The algorithm of Calcavecchia et al., named Backward Speculative Placement (BSP), is slightly different from the scheme of Algorithm 1, because it tentatively checks the migration of all VMs of the most loaded PM to all other PMs, as opposed to greedily selecting the first appropriate PM [17]. From the possible migrations, it selects the one which, based on historic workload data, leads to the smallest risk of demand dissatisfaction, using a scoring mechanism called "demand risk."

3.7 Evaluation

The last step in the envisioned experimentation process is to collect the appropriate metrics from the simulation runs and evaluate the algorithms based on these metrics.

CloudSim does a good job in logging all important events of a simulation and in calculating several aggregated performance metrics for each simulation run. These metrics include

- Total energy consumption
- SLA violation time per active host (SLATAH), showing the average percentage of time in which host utilization was 100%. The intuition behind this metric is that the saturation of resources indicates that VMs do not receive the required amount of resources, leading to SLA violations.
- Performance degradation due to migrations (PDM), showing the average performance degradation, relative to host capacity, caused by migrations. For this calculation, CloudSim assumes that the performance of a VM degrades by 10% of its CPU utilization during migration [6].

We extended CloudSim to write these metrics into a structured file (in tab-separated-values format), which we can then post-process and analyze using a standard spreadsheet program. In our analysis, we use the above three metrics to characterize the performance of the algorithms. This allows us to obtain a clear picture about the strengths and weaknesses of each algorithm (e.g., one algorithm may provide low energy consumption, but at the cost of many overloads, while for another algorithm it can be vice versa). In general, we can assume that different algorithms realize different trade-offs among these metrics.

4 Comparison

In our experiments, we compared the 7 algorithms presented in Section 3.6, using the tools and methodology outlined above.

4.1 Test configurations

As a baseline configuration, we reused the configuration defined by Beloglazov et al. [6]. Details of this configuration are shown in Table 3.

In each experiment, we aimed at investigating the effect of a different aspect on the effectiveness of the algorithms. Table 4 shows an overview of the tested aspects and the set of test cases for each

Table 3: Baseline configuration, defined by Beloglazov et al. [6]

VMs' requested CPU sizes [MIPS]:	2500, 2000, 1000, 500
PM CPU capacities [MIPS]:	1860, 2660
PM power models:	HP ProLiant ML110 G4, HP ProLiant ML110 G5
Workload:	PlanetLab trace
Overload detection:	Local regression

Table 4: Overview of the performed experiments

Aspect	Test cases
Workload	PlanetLab Bitbrains Google cluster Periodic (all VMs' load changes with the same periodicity) Markov-modulated Poisson
CPU load of VMs	constant 10% of requested capacity constant 50% of requested capacity constant 100% of requested capacity
VM size	Smaller than in the baseline configuration According to the baseline configuration Bigger than in the baseline configuration
PM capacity	All PMs have the same capacity 2 different PM types with differing capacity 5 different PM types with differing capacity
PM power characteristics	All PMs have the same power characteristics Multiple different power characteristics Power characteristics with small slope Power characteristics with large slope
Host overload detection	Static threshold Local regression

tested aspect; each of these tests are explained in more detail in the corresponding subsections of Section 4.2. In each experiment, only the given aspect is changed from the baseline configuration, so that the effects of each aspect can be investigated in isolation. This approach also allowed us to keep the number of experiments manageable: had we decided to test all possible combinations of these aspects, this would have lead to an explosion of test results, making it harder to extract the effect of the individual aspects.

4.2 Test results

In the following, we present the results of all experiments in tabular form. For each test case (i.e., for each row in the following tables), the best result is marked bold, as well as any other results within 5% of the best one. In each case, smaller numbers are better.

Algorithm runtimes are not reported because all investigated algorithms were very fast: all runtimes were well below one second on a PC with Intel Core I3-3110M processor running at 2.4 GHz.

Table 5: Effect of different workload traces on energy consumption [kWh]

Workload	Beloglazov	Lago	Chowdhury	Guazzone	Shi – PU	Shi – AC	Calcavecchia
PlanetLab	151.37	154.87	134.38	**115.29**	**117.85**	**115.72**	126.80
Bitbrains	152.37	150.06	**132.04**	**130.20**	**130.21**	**130.29**	**136.46**
Google cluster	70.40	69.60	65.24	**59.82**	66.71	**59.88**	63.37
Periodic	710.67	679.90	673.37	**533.55**	**556.49**	**533.35**	598.75
Markov	201.77	205.21	162.85	**142.06**	**146.28**	**142.42**	156.74

Table 6: Effect of different workload traces on SLATAH (SLA violation time per active host)

Workload	Beloglazov	Lago	Chowdhury	Guazzone	Shi – PU	Shi – AC	Calcavecchia
PlanetLab	7.07%	7.00%	5.10%	**4.05%**	5.00%	**3.86%**	4.11%
Bitbrains	4.30%	3.54%	**1.58%**	1.80%	2.18%	1.77%	2.22%
Google cluster	12.18%	11.56%	11.04%	11.76%	**10.36%**	11.91%	**10.82**%
Periodic	4.24%	**3.72%**	13.95%	4.16%	5.69%	4.16%	11.83%
Markov	10.13%	**8.98%**	9.84%	**8.67%**	9.72%	**8.55%**	9.55%

4.2.1 Different workload traces

The effect of different workload traces on the three investigated metrics are shown in Tables 5-7. The data reveal some interesting facts:

- Concerning energy consumption, the algorithm of Guazzone and the two algorithms of Shi deliver quite consistently the best results. Also consistently, the algorithms of Beloglazov and Lago deliver the highest costs, whereas the algorithms of Chowdhury and Calcavecchia are somewhere in the middle. Interestingly, these clusters and their order remain stable across the different workloads. The only variability that can be observed relates to the algorithms of Chowdhury and Calcavecchia: their performance both relative to each other and to the other clusters shows some oscillation. But the overall impact of the workload on the order of the algorithms' results is smaller than expected.

- There is a difference of up to 44% in total energy consumption between the best and the worst performing algorithm.

- Concerning SLATAH, the results are rather inconclusive. Interestingly, the algorithms that perform well relating to energy consumption, often have also good SLATAH values (e.g., Guazzone and Shi – AC) and algorithms with high energy consumption also have high SLATAH (especially Beloglazov – the algorithm of Lago delivers good SLATAH on the artificial workloads but worse results on the real-world traces).

- Concerning PDM, again the algorithm of Guazzone and the two algorithms of Shi deliver consistently the best results. The worst result is up to 4 times as high as the best one.

Table 7: Effect of different workload traces on PDM (performance degradation due to migrations)

Workload	Beloglazov	Lago	Chowdhury	Guazzone	Shi – PU	Shi – AC	Calcavecchia
PlanetLab	0.10%	0.13%	0.05%	**0.04%**	**0.04%**	**0.04%**	0.06%
Bitbrains	0.08%	0.07%	0.03%	**0.02%**	0.03%	0.03%	0.04%
Google cluster	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**
Periodic	0.09%	0.09%	0.10%	**0.07%**	**0.07%**	**0.07%**	0.11%
Markov	0.13%	0.17%	0.10%	**0.08%**	**0.08%**	**0.08%**	0.12%

Contrary to our expectation, it is not true that an algorithm that performs well on one metric would necessarily perform poorly on the other metrics. Rather, we have a cluster of three algorithms that perform clearly better than the other algorithms on two metrics, without a noticeable disadvantage on the third metric.

4.2.2 CPU load of VMs

In this test, we used a time-independent workload for each VM, which was specified as a given percentage of the VM's requested capacity. Specifically, we tested 10%, 50%, and 100% of the requested capacity. In order to save space, we present only the energy consumption and SLA violation tables for this and the following experiments. (But the full set of result data is available from the repository mentioned in Section 4.3.)

Table 8: Effect of the VMs' CPU load on energy consumption [kWh]

CPU load	Beloglazov	Lago	Chowdhury	Guazzone	Shi – PU	Shi – AC	Calcavecchia
10%	126.10	126.30	**119.07**	**116.25**	**116.25**	**116.25**	**121.06**
50%	632.01	**595.61**	**594.63**	**576.16**	**604.93**	**576.16**	635.39
100%	1471.44	**1207.51**	1464.35	**1202.62**	1509.92	**1202.62**	1506.87

Table 9: Effect of the VMs' CPU load on SLATAH (SLA violation time per active host)

CPU load	Beloglazov	Lago	Chowdhury	Guazzone	Shi – PU	Shi – AC	Calcavecchia
10%	1.46%	1.14%	0.05%	**0.03%**	**0.03%**	**0.03%**	0.04%
50%	0.12%	0.07%	**0.03%**	**0.03%**	**0.03%**	**0.03%**	**0.03%**
100%	0.02%	**0.00%**	**0.00%**	0.01%	**0.00%**	0.01%	**0.00%**

The results of this experiment are shown in Tables 8 (energy consumption) and 9 (SLA violations). Compared to the previous experiment, the differences between the performance of the algorithms are now much lower, and 5 of the 7 investigated algorithms return near-minimal results. This is no surprise since in these test cases, the VMs' load is constant (i.e., not changing with time), which significantly limits the optimization possibilities for the placement algorithms. This also explains why the SLATAH values are much lower here than in the previous experiment.

It is important to note the changes in the order of the algorithms' results. For example, Lago now performs significantly better than Beloglazov, unlike in the previous experiment where they performed very similarly. Since the previous experiment used realistic workload traces, whereas this one uses constant – and hence unrealistic – workload, this finding shows how dangerous it is to extrapolate algorithm effectiveness from an experiment with unrealistic workload to realistic ones. (That is, based on this experiment one would be tempted to consider Lago clearly better than Beloglazov, but as we have already seen, this is not true for realistic workloads.)

4.2.3 VM size

Table 10: Effect of the VMs' size on energy consumption [kWh]

CPU size	Beloglazov	Lago	Chowdhury	Guazzone	Shi – PU	Shi – AC	Calcavecchia
Small	134.52	137.13	120.60	**99.75**	108.18	**100.18**	110.13
Baseline	151.37	154.87	134.38	**115.29**	**117.85**	**115.72**	126.80
Big	168.40	175.56	153.79	**134.00**	141.14	**134.55**	147.94

Table 11: Effect of the VMs' size on SLATAH (SLA violation time per active host)

CPU load	Beloglazov	Lago	Chowdhury	Guazzone	Shi – PU	Shi – AC	Calcavecchia
Small	6.95%	6.93%	3.65%	3.75%	4.31%	3.68%	**3.26%**
Baseline	7.07%	7.00%	5.10%	**4.05%**	5.00%	**3.86%**	4.11%
Big	6.87%	6.77%	4.54%	4.05%	4.58%	3.96%	**3.64%**

In this experiment, we slightly changed the requested VM sizes compared to the baseline configuration. Recall that in the baseline configuration, the requested VMs have sizes of 2500, 2000, 1000, and 500 MIPS. In the "Small" test case, we changed this to 2200, 1800, 800, and 400 MIPS; in the "Big" test case, we changed it to 2500, 2100, 1500, and 800 MIPS. Since the workload is given as percentage with respect to the requested size, these changes directly translate into smaller and bigger actual VM sizes, respectively.

The results on energy consumption are shown in Table 10. As expected, the energy consumption grows with increasing VM size for each algorithm. The effect is similar on each algorithm, so that their order is hardly affected. The algorithm "Shi – PU" performs slightly worse than "Shi – AC" (actually, the same applies to most of the previous tests as well, although the difference is typically marginal), which pushes its result a little bit above the 5% limit for both the "Small" and the "Big" test case. However, the clustered structure of the algorithms' performance can still be observed.

As shown in Table 11, the effect of the VMs' size on SLA violations is rather small and lacks an easily recognizable pattern. There are some slight changes in the order of the algorithms, but the clustered structure of the algorithms' performance remains also here.

Altogether, this experiment did not provide much new insight. The consequence for future similar studies is that it is safe to use a fixed set of requested VM sizes: the results gained this way will probably also hold for other requested VM sizes.

4.2.4 PM capacity

Table 12: Effect of the PM sizes' heterogeneity on energy consumption [kWh]

PM sizes	Beloglazov	Lago	Chowdhury	Guazzone	Shi – PU	Shi – AC	Calcavecchia
1	112.74	125.16	**98.86**	**98.33**	**98.54**	**98.85**	**99.55**
2	151.37	154.87	134.38	**115.29**	**117.85**	**115.72**	126.80
5	170.61	136.19	148.95	**102.46**	118.32	**102.61**	129.25

Table 13: Effect of the PM sizes' heterogeneity on SLATAH (SLA violation time per active host)

PM sizes	Beloglazov	Lago	Chowdhury	Guazzone	Shi – PU	Shi – AC	Calcavecchia
1	5.18%	5.77%	**3.06%**	4.04%	3.94%	4.06%	**3.04%**
2	7.07%	7.00%	5.10%	**4.05%**	5.00%	**3.86%**	4.11%
5	8.86%	6.88%	**3.72%**	3.80%	4.51%	**3.60%**	4.87%

In this experiment, we test the effects of the heterogeneity of the PMs' sizes (capacities). In each test case, 800 PMs are available. In the first test case, each PM has capacity 2700. In the second test case, which is the baseline configuration, half of the PMs has capacity 1860, the other half has capacity 2660. In the last test case, five PM types are used with capacities 1500, 1860, 2000, 2660, and 3000, and there are equal number of PMs of each type.

The results on energy consumption are shown in Table 12. As can be seen, the difference between the algorithms' results is smallest in the homogeneous case: several algorithms achieve results very near to the best one, and the worst result is only 27% worse than the best. With increasing heterogeneity of the PMs' capacity, the differences between the performance of the algorithms grow: with five different PM types, the algorithms of Guazzone and "Shi − AC" emerge as clear winners, with the worst results being over 66% higher than theirs.

The results on SLA violations, shown in Table 13, are similar (although more noisy) to the energy consumption results. Also here, the difference between the worst and best result is significantly larger for the most heterogeneous case (146%) than in the two other cases (90% and 83%). For some algorithms (e.g., Beloglazov), increasing capacity heterogeneity leads to an increasing number of SLA violations, whereas for others (e.g., Shi − AC), exactly the opposite is true. Remarkably, the algorithms Guazzone and Shi − AC perform very well for the highly heterogeneous case in terms of both energy consumption and SLA violations at the same time.

PM size heterogeneity is a feature that a VM placement algorithm can take advantage of by preferring PMs with higher capacity. From the results it is evident that some algorithms do this better than others. For example, the algorithm of Lago performs relatively poorly for homogeneous PM sizes, but with growing PM size heterogeneity, its results are becoming better relative to the others. This is consistent with the findings of Lago et al. [44].

4.2.5 PM power characteristics

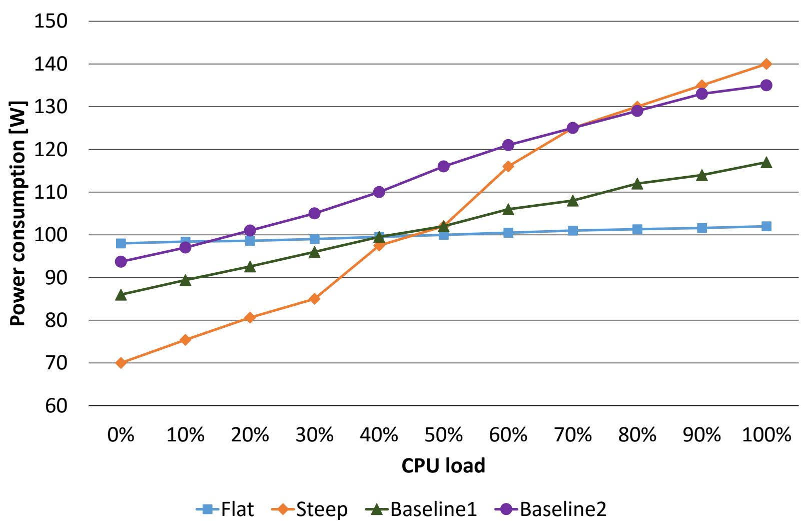


Figure 8: Different power consumption characteristics of PMs used in the "PM power characteristics" experiment

In this experiment, we vary the power consumption characteristics of the PMs, i.e., how their power consumption depends on their CPU load. In the baseline configuration, two kinds of PMs are used with different power characteristics, shown as Baseline1 and Baseline2 in Figure 8. Now we introduce three further configurations[8]. In the configurations termed Flat and Steep, all PMs share the same power characteristic, which is flatter or steeper than in the baseline, respectively, as shown in Figure 8. Finally, the Mixed configuration uses three different power characteristics, namely Baseline1, Flat, and Steep (with one third of the PMs having each characteristic).

The corresponding results on energy consumption are shown in Table 14. As can be seen, the Flat and Steep configurations lead to similar relative results as the Baseline configuration. We attribute this to the homogeneity of the power characteristics of all PMs in the Flat and Steep configurations, allowing no extra optimization. In the baseline configuration, there are

[8]The two baseline power characteristics model real-world servers. The newly introduced power characteristics are fictive: they were chosen so as to be able to study the effect of different power efficiencies, while at the same time having on average approximately the same power consumption as the baseline to foster comparability

Table 14: Effect of PM power characteristics on energy consumption [kWh]

Power models	Beloglazov	Lago	Chowdhury	Guazzone	Shi – PU	Shi – AC	Calcavecchia
Baseline	151.37	154.87	134.38	**115.29**	**117.85**	**115.72**	126.80
Flat	104.91	117.64	112.31	**91.14**	**94.71**	**91.57**	106.97
Steep	125.90	133.39	142.50	**115.41**	**119.65**	**115.78**	134.43
Mix	**104.21**	115.26	144.15	115.15	107.68	**102.09**	120.06

Table 15: Effect of PM power characteristics on SLATAH (SLA violation time per active host)

Power model	Beloglazov	Lago	Chowdhury	Guazzone	Shi – PU	Shi – AC	Calcavecchia
Baseline	7.07%	7.00%	5.10%	**4.05%**	5.00%	**3.86%**	4.11%
Flat	5.49%	6.17%	4.21%	**4.05%**	5.00%	**3.86%**	4.11%
Steep	5.51%	6.13%	4.15%	**4.05%**	5.00%	**3.86%**	4.11%
Mix	5.58%	6.17%	4.19%	**4.00%**	5.00%	**3.86%**	4.11%

two different power characteristics, introducing some power consumption heterogeneity, but their slopes are very similar, so that the difference in power efficiency is rather small.

However, the Mix configuration introduces a new level of power consumption heterogeneity because it contains PMs with significantly differing power efficiencies. As a result, the order of the algorithms' performance changes considerably. The algorithm "Shi – AC" continues to deliver excellent results, but the other high-performer so far, Guazzone, fails to exploit the opportunities inherent in heterogeneous power efficiency and returns results with almost 13% higher costs than the minimum. In contrast, Beloglazov's algorithm shows excellent performance on this test case, unlike in most test cases so far. On one hand, this is not surprising since this algorithm explicitly considers power efficiency when selecting the next PM; on the other hand, it is interesting to note that this excellent performance was not at all visible in the baseline configuration – which was also defined by Beloglazov et al. – where power efficiencies hardly differed. It can also be seen that Chowdhury's algorithm, the PM selection of which is the opposite from Beloglazov's, performs poorly on this test case, with costs 41% higher than the minimum.

It is important to observe that this test case – heterogeneous PM power characteristics – is the only one that completely disrupts the algorithm clusters and their order which seemed to be quite consistent in the previous experiments.

Concerning SLA violations, shown in Table 15, the impact of the power consumption characteristics is clearly much smaller. Most algorithms do not show any sensitivity in this respect.

4.2.6 Host overload detection

Table 16: Effect of the host overload detection technique on energy consumption [kWh]

Overload detection	Beloglazov	Lago	Chowdhury	Guazzone	Shi – PU	Shi – AC	Calcavecchia
Static threshold	183.61	176.42	139.49	**118.97**	**122.83**	**119.05**	131.05
Local regression	151.37	154.87	134.38	**115.29**	**117.85**	**115.72**	126.80

In this experiment, a static and an adaptive overload detection method are compared. The static method simply considers a PM overloaded if its load is higher than a predefined threshold. The adaptive method considers beside the current load level also the temporal development of the load: based on the last couple of measurements, it can make a prediction about the anticipated load and decide on this basis whether the PM should be considered overloaded. In the terminology of [49], the second approach integrates *load prediction* into VM placement.

Table 17: Effect of the host overload detection technique on SLATAH (SLA violation time per active host)

Power model	Beloglazov	Lago	Chowdhury	Guazzone	Shi – PU	Shi – AC	Calcavecchia
Static threshold	6.84%	6.84%	5.14%	**4.45%**	4.80%	**4.46%**	4.90%
Local regression	7.07%	7.00%	5.10%	**4.05%**	5.00%	**3.86%**	4.11%

Beloglazov et al. experimented with several adaptive overload detection methods and reported that, for their algorithm, local regression leads to the best results – and, in particular, better results than the static approach [6]. Local regression works by interpolating a polynomial function on the last couple of measured values and then using this polynomial for predicting the next value. In this case, the last 10 measurements of the CPU load are used to interpolate a linear function, this linear function is used to predict the CPU load for the next period, which in turn is used to determine whether the PM should be considered overloaded.

In Table 16, it can be seen that, in terms of energy consumption, actually *all* algorithms benefit from load prediction. However, it is interesting to note that the relative improvement of the algorithms can be quite different: in particular, the results of the best-performing algorithms improve only marginally (e.g., 3% in the case of Guazzone), whereas the improvement is much larger for the less successful algorithms (e.g., 18% in the case of Beloglazov). A possible explanation is that the best algorithms per se react quickly to the important changes in the workload, whereas other algorithms react more slowly and hence benefit from load prediction. Interestingly, the three best algorithms yield significantly better results even without load prediction than the remaining four algorithms with load prediction.

Another interesting consequence of the above phenomenon is that the use of local regression decreases the variability in the algorithms' results: while the difference between the best and worst result is 54% with the static threshold, the use of local regression decreases it to 34%. However, the clusters of algorithms and their order does not change.

In terms of SLA violations, the effect of load prediction is less pronounced. As shown in Table 17, some algorithms (e.g., Calcavecchia) benefit from load prediction, whereas others (e.g., Beloglazov) suffer a slight degradation. Interestingly, the effect of load prediction on SLA violations seems to be inverse to its effect on energy consumption: it tends to further improve the algorithms that perform well even without load prediction, while slightly deteriorating the less effective algorithms. Thus, load prediction actually increases the difference between the best and worst result in terms of SLATAH from 54% to 83%. Nevertheless, the order of the algorithms is hardly affected.

4.3 Availability

All our source code, configuration files, and results are publicly available from `https://github.com/zoltanmann/vm-alloc-comparison`.

5 Conclusions

5.1 Summary of results

In this paper, we presented an environment for experimentally evaluating and comparing the performance of VM placement algorithms. The environment, which is publicly available, builds on the popular open-source CloudSim toolkit and extends it with a standardized input data format, converters for publicly available workload traces, and workload generation facilities.

Using this evaluation environment, we performed a thorough comparison of 7 algorithms that solve the same version of the VM placement problem. The main findings of this evaluation are:

- Contrary to our expectation, it is not true that an algorithm that performs well on one metric would perform poorly on the other metrics. Rather, we have a cluster of three algorithms that perform clearly better than the other algorithms on two metrics, without a noticeable disadvantage on the third metric.
- The energy consumption of the worst result can be as much as 66% higher than the minimum. The differences with respect to the other metrics can be even much higher.
- With growing heterogeneity of the PMs – with respect to both capacity and power efficiency – the differences between the algorithms' performance in terms of energy consumption also increases. In terms of SLA violations, capacity heterogeneity also leads to a wider gap between the best and worst performing algorithm (but the heterogeneity of PM power consumption characteristics does not show a similar effect on SLA violations).
- The heterogeneity of the PMs also significantly influences the performance of the algorithms relative to each other. In contrast, different workload patterns had less influence on the relative performance of the algorithms.
- Load prediction improves the performance of all algorithms, but it primarily does so for the weaker algorithms. The best algorithms yield significantly better results even without load prediction than the remaining algorithms with load prediction.
- Although there is no clear winner, but generally the algorithms "Guazzone" and "Shi – AC" gave the best results.

5.2 Consequences

Based on these findings, several conclusions arise for VM placement researchers and practitioners:

- The big differences between the algorithms' performance highlight the importance of thorough empirical studies. Instead of comparing a new algorithm against trivial heuristics, as has been often done in the literature, a comparison with real competitors is much more meaningful. An open competition for VM placement algorithms would further foster the development of high-quality algorithms.
- Simplifying assumptions made by algorithm designers, such as homogeneity of PMs or ignorance of PMs' power consumption characteristics degrade algorithm performance in realistic settings. In particular, the heterogeneity of PMs in terms of capacity and power efficiency need to be taken into account when designing a VM placement algorithm.
- Which algorithm is the most suitable for a given data center depends on both the available hardware capabilities and the characteristics of the requested VMs. In particular, our findings show that the capabilities of the PMs in the data center play a key role – they seem to be even more important than the VMs' typical load patterns. Existing research (both algorithm design and evaluation) mostly focused on workload characteristics, but our results reveal that PM characteristics are at least as important, so that more research effort will be needed here.

5.3 Barriers to practical adoption

The investigated algorithms constitute academic work, and while working with them, we identified several aspects that need to be addressed before these algorithms can be applied in practical settings:

- SLA handling is limited to minimizing the time during which PMs are overloaded. In reality, SLAs can be much more sophisticated (e.g., penalties may depend on the length of continuous SLO violations, or on user-level metrics like response time).

- Most algorithms do not support VM properties like priorities, anti-colocation constraints, or the need for PMs with special hardware or software.
- Most algorithms assume that there are enough PMs to host all VMs. In practice, this may not always be the case, and hence VM placement algorithms need policies to handle over-subscription situations, for example, by temporarily pausing some VMs.
- Non-trivial interactions between VMs (e.g., the noisy neighbor phenomenon) or on the PM level (e.g., overheating) are not considered.

5.4 Future work

Concerning our own work, also several further research directions are promising. These include the investigation of other types of algorithms (e.g., population-based meta-heuristics) and other problem variants (e.g., considering multiple resource types or data transfer among VMs).

Acknowledgments

This work was partially supported by the Hungarian Scientific Research Fund (Grant Nr. OTKA 108947) and by the European Community's 7th Framework Programme (FP7/2007-2013) under grant 610802 (CloudWave).

References

[1] Ehsan Ahvar, Shohreh Ahvar, Zoltán Ádám Mann, Noel Crespi, Joaquin Garcia-Alfaro, and Roch Glitho. CACEV: a cost and carbon emission-efficient virtual machine placement method for green distributed clouds. In *Proceedings of the 13th IEEE International Conference on Services Computing*, pages 275–282, 2016.

[2] Alexandru-Florian Antonescu and Torsten Braun. SLA-driven simulation of multi-tenant scalable cloud-distributed enterprise information systems. In Florin Pop and Maria Potop-Butucaru, editors, *Adaptive Resource Management and Scheduling for Cloud Computing*, pages 91–102. Springer, 2014.

[3] Dávid Bartók and Zoltán Ádám Mann. A branch-and-bound approach to virtual machine placement. In *Proceedings of the 3rd HPI Cloud Symposium "Operating the Cloud"*, pages 49–63, 2015.

[4] D. M. Batista, N. L. S. da Fonseca, and F. K. Miyazawa. A set of schedulers for grid networks. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, pages 209–213, 2007.

[5] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28:755–768, 2012.

[6] Anton Beloglazov and Rajkumar Buyya. 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.

[7] Anton Beloglazov and Rajkumar Buyya. Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. *IEEE Transactions on Parallel and Distributed Systems*, 24(7):1366–1379, 2013.

[8] Ofer Biran, Antonio Corradi, Mario Fanelli, Luca Foschini, Alexander Nus, Danny Raz, and Ezra Silvera. A stable network-aware VM placement for cloud systems. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 498–506, 2012.

[9] Luiz F. Bittencourt, Edmundo R.M. Madeira, and Nelson L.S. da Fonseca. Scheduling in hybrid clouds. *IEEE Communications Magazine*, 50(9):42–47, 2012.

[10] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing SLA violations. In *10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 119–128, 2007.

[11] Dejene Boru, Dzmitry Kliazovich, Fabrizio Granelli, Pascal Bouvry, and Albert Y. Zomaya. Energy-efficient data replication in cloud computing datacenters. *Cluster Computing*, 18(1):385–402, 2015.

[12] R. v. d. Bossche, K. Vanmechelen, and J. Broeckhove. Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In *IEEE 3rd International Conference on Cloud Computing*, pages 228–235, 2010.

[13] D. Breitgand and A. Epstein. SLA-aware placement of multi-virtual machine elastic services in compute clouds. In *12th IFIP/IEEE International Symposium on Integrated Network Management*, pages 161–168, 2011.

[14] David Breitgand and Amir Epstein. Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds. In *Proceedings of IEEE Infocom*, pages 2861–2865, 2012.

[15] Dario Bruneo. A stochastic model to investigate data center performance and QoS in IaaS cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):560–569, 2014.

[16] Rajkumar Buyya and Manzur Murshed. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[17] Nicolo Maria Calcavecchia, Ofer Biran, Erez Hadad, and Yosef Moatti. VM placement strategies for cloud scenarios. In *IEEE 5th International Conference on Cloud Computing*, pages 852–859, 2012.

[18] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. 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.

[19] Emiliano Casalicchio, Daniel A. Menascé, and Arwa Aldhalaan. Autonomic resource provisioning in cloud systems with availability goals. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, page Article 1, 2013.

[20] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, 2014.

[21] Ming Chen, Hui Zhang, Ya-Yunn Su, Xiaorui Wang, Guofei Jiang, and Kenji Yoshihira. Effective VM sizing in virtualized data centers. In *IFIP/IEEE International Symposium on Integrated Network Management*, pages 594–601, 2011.

[22] Mohammed Rashid Chowdhury, Mohammad Raihan Mahmud, and Rashedur M. Rahman. Study and performance analysis of various VM placement strategies. In *16th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, page DOI: 10.1109/SNPD.2015.7176234, 2015.

[23] Rajarshi Das, Jeffrey O. Kephart, Charles Lefurgy, Gerald Tesauro, David W. Levine, and Hoi Chan. Autonomic multi-agent management of power and performance in data centers. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track*, pages 107–114, 2008.

[24] Dinil Mon Divakaran, Tho Ngoc Le, and Mohan Gurusamy. An online integrated resource allocator for guaranteed performance in data centers. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1382–1392, 2014.

[25] Mohamed Esam Elsaid and Christoph Meinel. Multiple virtual machines live migration performance modelling – VMware vMotion based study. In *Proceedings of the IEEE International Conference on Cloud Engineering*, pages 212–213, 2016.

[26] Michael Galloway, Gabriel Loewen, and Susan Vrbsky. Performance metrics of virtual machine live migration. In *Proceedings of the 8th IEEE International Conference on Cloud Computing*, pages 637–644, 2015.

[27] Rajeshwari Ganesan, Santonu Sarkar, and Akshay Narayan. Analysis of SaaS business platform workloads for sizing and collocation. In *IEEE 5th International Conference on Cloud Computing*, pages 868–875, 2012.

[28] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, 79:1230–1242, 2013.

[29] Saurabh Kumar Garg and Rajkumar Buyya. NetworkCloudSim: Modelling parallel applications in cloud simulations. In *Proceedings of the 4th IEEE/ACM International Conference on Utility and Cloud Computing*, pages 105–113, 2011.

[30] Rahul Ghosh and Vijay K. Naik. Biting off safely more than you can chew: Predictive analytics for resource over-commit in IaaS cloud. In *IEEE 5th International Conference on Cloud Computing*, pages 25–32, 2012.

[31] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Resource pool management: Reactive versus proactive or let's be friends. *Computer Networks*, 53(17):2905–2922, 2009.

[32] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, Guillaume Belrose, Tom Turicchi, and Alfons Kemper. An integrated approach to resource pool management: Policies, efficiency and quality metrics. In *IEEE International Conference on Dependable Systems and Networks*, pages 326–335, 2008.

[33] M. Guazzone, C. Anglano, and M. Canonico. Exploiting VM migration for the automated power and performance management of green cloud computing systems. In *1st International Workshop on Energy Efficient Data Centers*, pages 81–92, 2012.

[34] Brian Guenter, Navendu Jain, and Charles Williams. Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In *Proceedings of IEEE INFOCOM*, pages 1332–1340, 2011.

[35] Sijin He, Li Guo, Moustafa Ghanem, and Yike Guo. Improving resource utilisation in the cloud environment using multivariate probabilistic models. In *IEEE 5th International Conference on Cloud Computing*, pages 574–581, 2012.

[36] Nguyen Trung Hieu, Mario Di Francesco, and Antti Ylä-Jääski. Virtual machine consolidation with usage prediction for energy-efficient cloud data centers. In *Proceedings of the 8th IEEE International Conference on Cloud Computing*, pages 750–757, 2015.

[37] Takahiro Hirofuchi, Adrien Lebre, and Laurent Pouilloux. SimGrid VM: Virtual machine support for a simulation framework of distributed systems. *IEEE Transactions on Cloud Computing*, page DOI: 10.1109/TCC.2015.2481422, 2016.

[38] Chris Hyser, Bret McKee, Rob Gardner, and Brian J. Watson. Autonomic virtual machine placement in the data center. Technical report, HP Laboratories, 2008.

[39] Deepal Jayasinghe, Calton Pu, Tamar Eilam, Malgorzata Steinder, Ian Whalley, and Ed Snible. Improving performance and availability of services hosted on IaaS clouds with structural constraint-aware virtual machine placement. In *IEEE International Conference on Services Computing*, pages 72–79, 2011.

[40] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu. Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In *IEEE 30th International Conference on Distributed Computing Systems*, pages 62–73, 2010.

[41] Gabor Kecskemeti. DISSECT-CF: A simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58:188–218, 2015.

[42] A. Khosravi, S. K. Garg, and R. Buyya. Energy and carbon-efficient placement of virtual machines in distributed cloud data centers. In *European Conference on Parallel Processing*, pages 317–328, 2013.

[43] Madhukar Korupolu, Aameek Singh, and Bhuvan Bamba. Coupled placement in modern data centers. In *IEEE International Symposium on Parallel and Distributed Processing*, pages 1–12, 2009.

[44] D. G. do Lago, E. R. M. Madeira, and L. F. Bittencourt. Power-aware virtual machine scheduling on clouds using active cooling control and DVFS. In *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*, page Article 2, 2011.

[45] Hongjian Li, Guofeng Zhu, Chengyuan Cui, Hong Tang, Yusheng Dou, and Chen He. Energy-efficient migration and consolidation algorithm of virtual machines in data centers for cloud computing. *Computing*, 98(3):303–317, 2016.

[46] Rui Li, Qinghua Zheng, Xiuqi Li, and Jie Wu. A novel multi-objective optimization scheme for rebalancing virtual machine placement. In *IEEE 9th International Conference on Cloud Computing*, page to appear, 2016.

[47] W. Li, J. Tordsson, and E. Elmroth. Virtual machine placement for predictable and time-constrained peak loads. In *Proceedings of the 8th International Conference on Economics of Grids, Clouds, Systems, and Services*, pages 120–134, 2011.

[48] Vincenzo De Maio, Gabor Kecskemeti, and Radu Prodan. An improved model for live migration in data centre simulators. In *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 527–530, 2016.

[49] Z. Á. Mann. Allocation of virtual machines in cloud data centers – a survey of problem models and optimization algorithms. *ACM Computing Surveys*, 48(1):Article 11, 2015.

[50] Zoltán Ádám Mann. Approximability of virtual machine allocation: much harder than bin packing. In *Proceedings of the 9th Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, pages 21–30, 2015.

[51] Zoltán Ádám Mann. A taxonomy for the virtual machine allocation problem. *International Journal of Mathematical Models and Methods in Applied Sciences*, 9:269–276, 2015.

[52] Zoltán Ádám Mann. Interplay of virtual machine selection and virtual machine placement. In *Proceedings of the 5th European Conference on Service-Oriented and Cloud Computing*, pages 137–151, 2016.

[53] Zoltán Ádám Mann. Multicore-aware virtual machine placement in cloud data centers. *IEEE Transactions on Computers*, 65(11):3357–3369, 2016.

[54] Antonio Marotta and Stefano Avallone. A simulated annealing based approach for power efficient virtual machines consolidation. In *Proceedings of the 8th IEEE International Conference on Cloud Computing*, pages 445–452, 2015.

[55] Tran Ngoc Minh, Thoai Nam, and Dick H. J. Epema. Parallel workload modeling with realistic characteristics. *IEEE Transactions on Parallel and Distributed Systems*, 25(8):2138–2148, 2014.

[56] Mayank Mishra and Umesh Bellur. De-fragmenting the cloud. In *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 511–520, 2016.

[57] Mayank Mishra and Anirudha Sahoo. On theory of VM placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach. In *IEEE International Conference on Cloud Computing*, pages 275–282, 2011.

[58] Natural Resources Defense Council. Scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers. http://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf, 2014.

[59] Junjie Ni, Yuanqiang Huang, Zhongzhi Luan, Juncheng Zhang, and Depei Qian. Virtual machine mapping policy based on load balancing in private cloud environment. In *International Conference on Cloud and Service Computing*, pages 292–295, 2011.

[60] KyoungSoo Park and Vivek S. Pai. CoMon: a mostly-scalable monitoring system for PlanetLab. *ACM SIGOPS Operating Systems Review*, 40(1):65–74, 2006.

[61] Alejandro Pelaez, Manish Parashar, and Andres Quiroz. Dynamic adaptation of policies using machine learning. In *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 501–510, 2016.

[62] Larry Peterson, Andy Bavier, Marc E. Fiuczynski, and Steve Muir. Experiences building PlanetLab. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, pages 351–366, 2006.

[63] Sareh Fotuhi Piraghaj, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, and Rajkumar Buyya. ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers. *Software: Practice and Experience*, page to appear, 2016.

[64] Fabio López Pires and Benjamin Baran. A virtual machine placement taxonomy. In *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 159–168, 2015.

[65] A. Radhakrishnan and V. Kavitha. Energy conservation in cloud data centers by minimizing virtual machines migration through artificial neural network. *Computing*, pages DOI: 10.1007/s00607–016–0499–4, 2016.

[66] Safraz Rampersaud and Daniel Grosu. Sharing-aware online algorithms for virtual machine packing in cloud environments. In *Proceedings of the 8th IEEE International Conference on Cloud Computing*, pages 718–725, 2015.

[67] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the 3rd ACM Symposium on Cloud Computing*, page Article 7, 2012.

[68] B. C. Ribas, R. M. Suguimoto, R. A. N. R. Montano, F. Silva, L. de Bona, and M. A. Castilho. On modelling virtual machine consolidation to pseudo-Boolean constraints. In *13th Ibero-American Conference on AI*, pages 361–370, 2012.

[69] Ivan Rodero, Hariharasudhan Viswanathan, Eun Kyung Lee, Marc Gamell, Dario Pompili, and Manish Parashar. Energy-efficient thermal-aware autonomic management of virtualized HPC cloud infrastructure. *Journal of Grid Computing*, 10(3):447–473, 2012.

[70] Thiago Teixeira Sá, Rodrigo N. Calheiros, and Danielo G. Gomes. CloudReports: An extensible simulation tool for energy-aware cloud computing environments. In Zaigham Mahmood, editor, *Cloud Computing*, pages 127–142. Springer, 2014.

[71] M. A. Salehi, P. R. Krishna, K. S. Deepak, and R. Buyya. Preemption-aware energy management in virtualized data centers. In *5th International Conference on Cloud Computing*, pages 844–851, 2012.

[72] Mina Sedaghat, Eddie Wadbro, John Wilkes, Sara De Luna, Oleg Seleznjev, and Erik Elmroth. DieHard: reliable scheduling to survive correlated failures in cloud data centers. In *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 52–59, 2016.

[73] Siqi Shen, Vincent van Beek, and Alexandru Iosup. Statistical characterization of business-critical workloads hosted in cloud datacenters. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 465–474, 2015.

[74] L. Shi, J. Furlong, and R. Wang. Empirical evaluation of vector bin packing algorithms for energy efficient data centers. In *IEEE Symposium on Computers and Communications*, pages 9–15, 2013.

[75] W. Song, Z. Xiao, Q. Chen, and H. Luo. Adaptive resource provisioning for the cloud using online bin packing. *IEEE Transactions on Computers*, 63(11):2647–2660, 2014.

[76] Shekhar Srikantaiah, Aman Kansal, and Feng Zhao. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, page Article 10, 2008.

[77] Anja Strunk. Costs of virtual machine live migration: A survey. In *8th IEEE World Congress on Services*, pages 323–329, 2012.

[78] P. Svärd, W. Li, E. Wadbro, J. Tordsson, and E. Elmroth. Continuous datacenter consolidation. In *IEEE 7th International Conference on Cloud Computing Technology and Science*, pages 387–396, 2015.

[79] Fei Teng, Lei Yu, Tianrui Li, Danting Deng, and Frdric Magoules. Energy efficiency of VM consolidation in IaaS clouds. *The Journal of Supercomputing*, pages DOI: 10.1007/s11227–016–1797–5, 2016.

[80] Wenhong Tian, Minxian Xu, Aiguo Chen, Guozhong Li, Xinyang Wang, and Yu Chen. Open-source simulators for cloud computing: Comparative study and challenging issues. *Simulation Modelling Practice and Theory*, 58:239–254, 2015.

[81] L. Tomás and J. Tordsson. An autonomic approach to risk-aware data center overbooking. *IEEE Transactions on Cloud Computing*, 2(3):292–305, 2014.

[82] A. Verma, P. Ahuja, and A. Neogi. pMapper: power and migration cost aware application placement in virtualized systems. In *Middleware 2008 – Proceedings of the ACM/IFIP/USENIX 9th International Middleware Conference*, pages 243–264, 2008.

[83] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari. Server workload analysis for power minimization using consolidation. In *Proceedings of the 2009 USENIX Annual Technical Conference*, pages 355–368, 2009.

[84] Akshat Verma, Gautam Kumar, Ricardo Koller, and Aritra Sen. CosMig: Modeling the impact of reconfiguration in a cloud. In *19th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 3–11, 2011.

[85] David Villegas, Athanasion Antoniou, Seyed Masoud Sadjadi, and Alexandru Iosup. An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds. In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 612–619, 2012.

[86] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923–2938, 2009.

[87] Zhen Xiao, Qi Chen, and Haipeng Luo. Automatic scaling of internet applications for cloud computing services. *IEEE Transactions on Computers*, 63(5):1111–1123, 2014.

[88] Zhen Xiao, Weijia Song, and Qi Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1107–1117, 2013.

[89] Jianwei Yin, Xingjian Lu, Xinkui Zhao, Hanwei Chen, and Xue Liu. BURSE: A bursty and self-similar workload generator for cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 26(3):668–680, 2015.

[90] Zhiming Zhang, Chan-Ching Hsu, and Morris Chang. CoolCloud: A practical dynamic virtual machine placement framework for energy aware data centers. In *Proceedings of the 8th IEEE International Conference on Cloud Computing*, pages 758–765, 2015.

[91] Qinghua Zheng, Rui Li, Xiuqi Li, and Jie Wu. A multi-objective biogeography-based optimization for virtual machine placement. In *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 687–696, 2015.

[92] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova. 1000 islands: an integrated approach to resource management for virtualized data centers. *Cluster Computing*, 12(1):45–57, 2009.