Cloud simulators in the implementation and evaluation of virtual machine placement algorithms[*]

Zoltán Ádám Mann
paluno – The Ruhr Institute for Software Technology,
University of Duisburg-Essen, Essen, Germany

Abstract

In recent years, many algorithms have been proposed for the optimized allocation of virtual machines in cloud data centers. Such algorithms are usually implemented and evaluated in a cloud simulator. This paper investigates the impact of the choice of cloud simulator on the implementation of the algorithms and on the evaluation results. In particular, we report our experiences with porting an algorithm and its evaluation framework from one simulator (CloudSim) to another (DISSECT-CF). Our findings include limitations in the design of the simulators as well as in existing algorithm implementations. Based on this experience, we propose architectural guidelines for the integration of virtual machine allocation algorithms into cloud simulators.

Keywords: Cloud computing, virtual machine placement, virtual machine consolidation, resource allocation, cloud simulator, CloudSim, DISSECT-CF

1 Introduction

The last couple of years have witnessed a steady rise in the adoption of cloud services [31]. However, the obvious advantages of the cloud are not for free: cloud data centers consume massive amounts of electrical energy, leading to ever increasing costs and environmental impact [9]. Therefore, the energy-efficient utilization of cloud data centers has become a very important research topic.

The biggest savings in power consumption can be achieved through the consolidation of the workload to a low number of servers and switching the remaining servers to a low-power state (e.g., sleep or hibernate). This is made possible by virtualization technology: a single Physical Machine (PM) can host multiple Virtual Machines (VMs), allowing good utilization of the available hardware resources. Moreover, live migration of VMs allows data center operators to react to changes in the workload: in times of low load, the VMs are consolidated to a few PMs and the remaining PMs can be switched to a low-power mode, whereas in times of high load, PMs can be switched on again so that the VMs can be spread across a higher number of PMs, thus improving quality of service [34].

This leads to the optimization problem of determining the best placement of the VMs on the PMs, given the capacity of the PMs and the resource needs of the VMs [23]. In recent years, a large number of algorithms have been suggested in the literature to address this problem [20].

In order to demonstrate the practical applicability of such algorithms, they are typically implemented and evaluated using a cloud simulator. In principle, it would be better to test them in large-scale real-world data centers, but this is generally not possible: most researchers do not have access to real data centers and even if they have, applying a placement algorithm that is still in an experimental state to a real data center would be dangerous. Moreover, cloud simulators allow easy experimentation with many different setups, e.g., with different types of PMs and VMs or with arbitrarily scaling their number.

One of the most widely used cloud simulators is CloudSim, which is available as open-source software [6, 32]. Beside providing a comprehensive model of the relevant cloud entities (data centers, PMs, VMs etc.), CloudSim has the big advantage that it comes with a built-in competitive VM placement algorithm due to Beloglazov et al. [4, 3]. What is more, CloudSim also contains all the test data, test drivers etc. that were used to evaluate the algorithm of Beloglazov et al. We will refer to this evaluation setup contained in CloudSim as the *PlanetLab experiment* because it is based on workload traces from the PlanetLab system [28]. The fact that the algorithm of Beloglazov et al. and the PlanetLab experiment are integrated in CloudSim is probably one of the reasons why many researchers who have suggested new algorithms for the VM allocation problem used the algorithm of Beloglazov et al. as a comparison baseline and demonstrated improvements using the PlanetLab experiment.

[*]This paper was published in *Software: Practice and Experience*, 48(7):1368-1389, 2018

On the other hand, CloudSim has also received some critiques, for example for its over-simplified model of I/O processing [26], limited communication models [25], inaccuracy of communication models [37] and lack of support for Quality of Service [15].

Recently, the simulator DISSECT-CF has been developed and proposed as an alternative to CloudSim [16]. Just like CloudSim, also DISSECT-CF implements all cloud infrastructure elements necessary for VM allocation algorithms, like PMs and VMs. Extensive experiments in [16] gave evidence for two important advantages of DISSECT-CF:

- The simulation results of DISSECT-CF in terms of finishing time and energy consumption are very accurate, with error of just around 1% in most cases. In particular, the simulation results are more accurate than those of CloudSim.
- DISSECT-CF is more efficient, leading to significantly better scalability of simulation time than in the case of CloudSim. In some cases, DISSECT-CF was 2800 times faster than CloudSim.

Based on these promising characteristics, the aim of the work presented in this paper was to assess the suitability of DISSECT-CF for the implementation and evaluation of VM placement algorithms. For this purpose, we planned to reproduce the built-in VM placement algorithm of CloudSim as well as the PlanetLab experiment within DISSECT-CF. By this, we wanted to answer the following research questions:

- Is DISSECT-CF also capable of simulating a VM placement algorithm, just as CloudSim?
- Are there significant differences in the way a VM placement algorithm is to be implemented in different cloud simulators?
- Are there significant differences in the evaluation results of the same VM placement algorithm using different cloud simulators?
- Is the evaluation of a VM placement algorithm significantly faster with DISSECT-CF than with CloudSim?

Our – as it turned out, naive – idea was that reproducing the built-in VM placement algorithm of CloudSim in DISSECT-CF would be easy, for three reasons. First, both simulators simulate the same real-world phenomena, namely the running of VMs on PMs and migration of VMs between PMs in an infrastructure cloud. Second, a preliminary view of the two simulators showed that they use comparable abstractions. Third, both simulators are implemented in Java. For these reasons, we were hoping that we could actually port most of the relevant program parts from CloudSim to DISSECT-CF, only having to adjust them to the interface of DISSECT-CF. We were also hoping that these adjustments would primarily concern only syntactic differences.

However, this quickly turned out to be a much more complicated undertaking. The biggest difficulties were:

- The philosophy of the two simulators is actually very different. CloudSim's design aims at ease of use, whereas DISSECT-CF aims at precise simulation of reality. This leads to subtle differences that have nevertheless considerable impact on the programming interface of the two simulators.
  - For instance, switching a PM on or off is instantaneous in CloudSim, but takes some – simulated – time in DISSECT-CF (just as in reality). Therefore, if the VM placement algorithm decides to switch on a new PM p and migrate a VM v to p, this can be achieved in CloudSim using two lines of code: first switch on p, then migrate v to p. In DISSECT-CF, however, when the call to switch on p returns, p is not on yet, so trying to migrate v to p would result in an error. Instead, we must schedule the call to migrate v to p to some future point in time when p is already on.
  - Another important example is that in CloudSim, a PM can be queried to find out its current power consumption, or even how much power it would consume if its load increased by a given amount. This is not possible in DISSECT-CF (just like it is not possible in reality). Instead, a separate EnergyMeter can be used to measure the actual energy consumption of a PM. This is sufficient for the simulation, but insufficient for a VM placement algorithm that wants to give preference to PMs with higher power efficiency. Of course, a PM's power model (i.e., the dependence of its power consumption on the load of its resources) is stored in a PM in DISSECT-CF as well, but not exposed through its public interface, and exposing it would contradict the philosophy of DISSECT-CF.
- The implementation of Beloglazov's algorithm makes use of some special features of CloudSim. For instance, it is possible in CloudSim to remove a VM from a PM and decide later on where to migrate it or not to migrate it at all by putting it back. This is not realistic and consequently also not supported by DISSECT-CF. Therefore, in DISSECT-CF, the tentative VM movements that the algorithm experiments with must be decoupled from the actual VM migrations.

- Beyond the above major differences, also some inconsistencies and suboptimal solutions in the implementation of the algorithm and of the PlanetLab experiment in CloudSim have been identified.

For these reasons, reproducing the VM placement algorithm of CloudSim in DISSECT-CF came close to re-implementing it, instead of simply porting it (as originally assumed).

The contributions of this paper can be summarized as follows:

- We describe our experiences with reproducing the algorithm of Beloglazov et al. and the PlanetLab experiment in DISSECT-CF.
  - We propose solutions to the problems mentioned above stemming from the differences between the two simulators.
  - We highlight some shortcomings of the existing CloudSim implementation.
  - Since DISSECT-CF provides a much more realistic environment, porting an algorithm from CloudSim to DISSECT-CF can be seen as an intermediate step after which it can be more easily ported to a real environment.
- We compare empirically the performance of the same algorithm on the same input data in the two simulators.
- Based on the experiences, we propose guidelines for the implementation of VM placement algorithms in cloud simulators in general.

Our findings show that the used simulator has significant impact on the implementation of the algorithm as well as the evaluation results. This impact can be reduced by engineering an appropriate loose coupling between the algorithm and the rest of the simulator.

The remainder of the paper is organized as follows. Related work is covered in Section 2. Section 3 describes various background information like the VM placement problem in general, the two used simulators, and the PlanetLab dataset. Section 4 presents how we ported the algorithm and the PlanetLab experiment from CloudSim to DISSECT-CF, the challenges that we faced and the design choices made to cope with the challenges. In Section 5, we present the insights that we have gathered about the existing implementation of the algorithm and the experiment in CloudSim. Experimental results comparing the two simulators are shown in Section 6, followed by a discussion of the findings in Section 7. Finally, Section 8 concludes the paper.

2 Related work

A huge number of papers have been devoted to new algorithms for VM placement [20]. Given the high computational complexity of the problem [22], most approaches proposed the use of different heuristic algorithms. Concerning the effectiveness of these algorithms, few papers applied theoretical analysis [2, 21], but the vast majority used an empirical evaluation.

Some authors used real systems for evaluating the performance of VM placement algorithms (see for example [10]). While this method has the highest fidelity, the available hardware limits the possibilities of experimentation: for instance, it is not possible to test the scalability of the algorithms to thousands of machines unless the authors have access to such a huge data center, which is not the case for most researchers. Therefore, most authors used simulation. Some authors created their own simulators (see for example [11]), but the design and development of a good simulator and the careful validation of its fidelity are challenging tasks on their own.

Hence it is no surprise that cloud simulators, and in particular CloudSim, became very popular among cloud researchers. Many VM placement algorithms have been implemented and evaluated using CloudSim, see e.g. [4, 17, 18, 30, 1]. CloudSim has also been used to compare different VM placement algorithms with each other [8, 24], as well as for evaluating other types of algorithms, e.g., mobile application scheduling [19].

There are, of course, also some alternatives to CloudSim. A similarly well established simulator is SimGrid [7]: 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 [14]. DISSECT-CF is another recent alternative that was chosen for this study because of its improved simulation speed and accuracy [16]. DCSim is similar to CloudSim, but with a focus on transactional, continuous workloads [36]. There are also simulators with special emphasis on the networking part of cloud computing, like GreenCloud [5]. ReliaCloud-NS is a simulator that focuses on the evaluation of the reliability of cloud systems [33]. A general overview and comparison of four open-source cloud simulators (CloudSim, GreenCloud, iCanCloud, CloudSched) was given by Tian et al. [35]. CloudSim also has been extended in several ways, e.g., to simulate DVFS (Dynamic Voltage and Frequency Scaling) [12], containers [29] or fog computing systems [13].

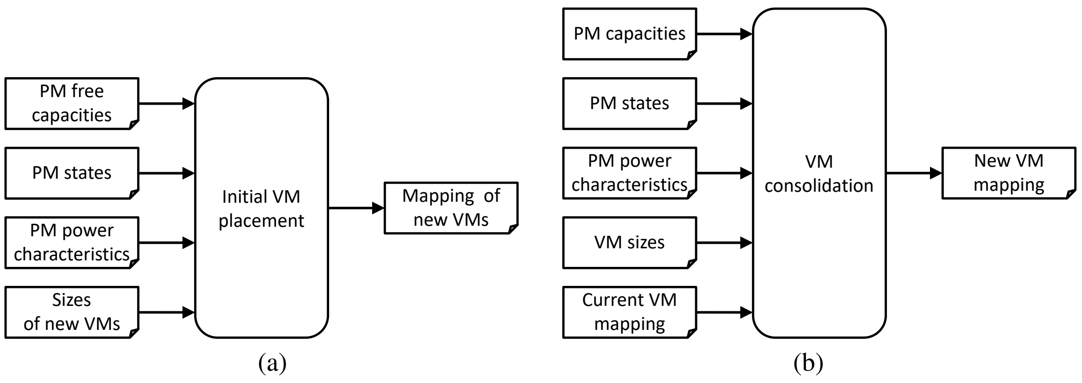

Figure 1: Inputs and outputs of (a) initial VM placement and (b) VM consolidation

To the best of our knowledge, the impact of the choice of cloud simulator on the implementation and evaluation of a VM placement algorithm has not been studied before, making our work novel. As explained in the Introduction, for our study we chose CloudSim as an established and popular simulator already containing an existing VM placement algorithm, and DISSECT-CF as a promising newer contestant that did not contain a sophisticated VM placement algorithm.

3 Background

3.1 VM placement

VM placement is about optimizing the mapping of VMs to PMs. It actually encompasses two different optimization problems:

- *Initial VM placement* is invoked when new VMs have been requested. The aim is to find, for each new VM, an appropriate PM that can host it.
- *VM consolidation* is regularly invoked to re-optimize the placement of existing VMs, with the aim to react to changes in the workload.

As shown in Fig. 1(a), initial VM placement expects multiple kinds of inputs: the free capacities, current state, and power characteristics of the available PMs, as well as the sizes of the newly requested VMs.[1] The free capacities of the PMs and the sizes of the VMs are with respect to a set of defined resource types (e.g., CPU, RAM, and disk). The state of a PM can be on or off. The power characteristic of a PM is a function that defines the PM's power consumption depending on its CPU load. The output of initial VM placement is a mapping of the new VMs to the available PMs, i.e., for each new VM it is decided which PM should host it.

In the case of VM consolidation (cf. Fig. 1(b)), inputs and outputs are similar but slightly different. In particular, the input contains the total capacity of the PMs and the size of all VMs, as well as the current mapping of VMs to PMs. The output is the new mapping of VMs to PMs. The implication of the new mapping is that, if a VM v is currently hosted by a PM p_1, but the new mapping foresees another PM p_2 for v, then v has to be migrated from p_1 to p_2.

The overall objectives of VM allocation are:

- Minimizing total energy consumption. This is primarily influenced by the average number of PMs that are turned on. Further, the difference in power efficiency among different PMs as well as the load-dependency of PMs' power consumption have to be taken into account.
- Minimizing the number of PM overloads. A PM overload occurs if the total resource demand of the VMs hosted by a PM surpasses the capacity of the PM for at least one resource type. PM overloads may lead to performance degradation and to violations of Service Level Objectives (SLOs) and should be thus avoided as much as possible.
- Minimizing the number of migrations. Since the migration of VMs incurs additional load on both servers and network resources, the number of migrations should be not too high.

[1]Both initial VM placement and VM consolidation have many different variants [20]. Here we define the variant that will be used throughout the paper.

It should be noted that these objectives are generally in conflict with each other, making VM allocation an interesting multi-objective optimization problem.

3.2 CloudSim

CloudSim was developed by researchers at the Cloud Computing and Distributed Systems Laboratory of the University of Melbourne [6]. 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 one of the most widely used cloud simulators [32]. In this work, version 4.0 of CloudSim was used. Over the years, CloudSim has gained a wide-ranging set of features, from which we now look at the ones that are most relevant for our purposes.

On the level of physical resources, CloudSim simulates data centers with an arbitrary number of PMs. Each PM has a given number of CPU cores (called processing elements or PEs in CloudSim's terminology) with given capacity, as well as a given amount of RAM, storage capacity, and network bandwidth. The power characteristic of a PM can be specified by means of a *power model*. CloudSim comes with multiple pre-defined power models, e.g., a parameterizable linear model as well as the measured power characteristics of some specific PM types (these are measured at discrete load values using the SPECpower benchmarks and linearly interpolated between the measurements).

Interestingly, power states (such as on and off) of the PMs are not modeled. As a consequence, also the bootup and shutdown processes – along with their duration and energy consumption – are not modeled. At some points, CloudSim assumes that a PM with CPU load 0 is switched off and consumes 0 power (e.g., the linear power model behaves this way, but the measurements-based power models do not).

Virtual machines are characterized by their requirements on the same set of resources as provided by PMs: CPU, RAM, storage, and network bandwidth. For each resource, the VM stores two values: the required and the allocated amount. The CPU is handled in a special way: beyond the number of cores, the required amount of per-core CPU capacity is specified as a scalar value, whereas the allocated amount is an array, the elements of which specify the allocated CPU capacity for each vCPU.

In contrast to PMs, the state of VMs is modeled, using two flags. One of them indicates whether the VM is in the process of being instantiated, the other shows whether the VM is being migrated. During the migration, CloudSim assumes that the source PM experiences 90% of the VM's CPU load, whereas the destination PM experiences 10%.

VMs can host so-called cloudlets, which model applications, tasks, or other pieces of software to be deployed. A cloudlet's resource utilization – in terms of CPU, RAM, and network bandwidth – can be defined with a utilization model. The utilization model specifies resource consumption as a function of time. CloudSim comes with several predefined utilization models, like constant 0, constant 100%, random, and file-based. The latter, which will be particularly important for our purposes (see Section 3.4), supports the usage of workload traces.

3.3 The algorithm of Beloglazov et al.

Beloglazov et al. described their algorithm in multiple papers [4, 3]. It is actually a pair of (strongly linked) heuristic algorithms: one for initial VM placement and one for VM consolidation.

For placing a new VM, the algorithm looks for a PM that has sufficient capacity to host the new VM. If there are multiple such PMs, then it selects the one where the new VM leads to the lowest increase in power consumption.

VM consolidation consists of the following steps:

1. The over-utilized PMs are determined.
2. For each over-utilized PM, some VMs to remove are determined.
3. The list of all VMs to migrate is sorted in non-increasing order of CPU size.
4. In this order, each VM is migrated to a new PM, where the new PM is determined using the algorithm for initial VM placement.
5. For all PMs that were not over-utilized, the algorithm tries to migrate all hosted VMs to other PMs with the same method as above. If this is possible for a PM, then all its VMs are indeed migrated; otherwise, none of the VMs of the PM are actually migrated.

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

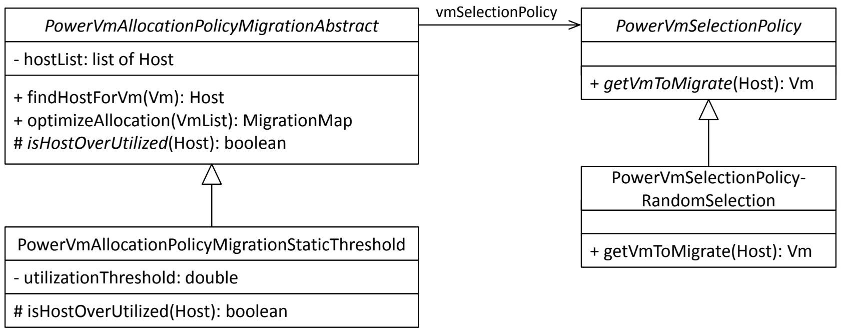
PowerVmAllocationPolicyMigrationAbstract
- hostList: list of Host
+ findHostForVm(Vm): Host
+ optimizeAllocation(VmList): MigrationMap
# isHostOverUtilized(Host): boolean
vmSelectionPolicy
PowerVmSelectionPolicy
+ getVmToMigrate(Host): Vm
PowerVmSelectionPolicy-RandomSelection
+ getVmToMigrate(Host): Vm
PowerVmAllocationPolicyMigrationStaticThreshold
- utilizationThreshold: double
# isHostOverUtilized(Host): boolean

Figure 2: Simplified class diagram showing the classes implementing the algorithm of Beloglazov et al. in CloudSim

For the first two steps of the algorithm, the authors experimented with several different variants. In step 1, different policies can be used to determine when a PM should be considered over-utilized. The easiest way is to use a static threshold, i.e., the PM is considered over-utilized if and only if its utilization is higher than a given threshold. However, more sophisticated mechanisms can be used to also take into account patterns and trends in the temporal evolution of the PM's utilization and predict over-utilization. In step 2, different policies can be used to determine which VM to remove from an over-utilized PM, e.g., based on the CPU and memory size of the available VMs.

The algorithm of Beloglazov et al. is implemented in CloudSim mainly in the class `PowerVmAllocationPolicyMigrationAbstra` (see Fig. 2). This abstract class contains most of the code of the algorithm; the subclasses of this class contain different policies for determining over-utilization of PMs in the form of different implementations of the `isHostOverUtilized` method. The decision on which VM to select from an over-utilized PM is delegated to a `PowerVmSelectionPolicy` object. Different subclasses of this class provide different implementations of the `getVmToMigrate` method for this purpose.

Although CloudSim contains several policies for determining over-utilization of PMs and for deciding which VM to select from an over-utilized PM, in this paper, we focus on one specific combination of these policies. Namely, over-utilization of a PM is decided based on whether the utilization of the PM exceeds a given threshold $0 < \tau \leq 1$. The VM to be migrated from an over-utilized PM is selected so as to minimize the migration time: for this reason, the VM with smallest RAM size is selected.

The entry points to the algorithm are within the class `PowerVmAllocationPolicyMigrationAbstract` the methods `findHostForVm` (for initial VM allocation) and `optimizeAllocation` (for VM consolidation). The class has roughly 30 further methods (not shown in Fig. 2) that are mostly called from `optimizeAllocation`. Furthermore, the algorithm also calls numerous methods in other classes, such as in `Vm` and `Host`.

3.4 The PlanetLab experiment

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 [28]. CoMon was a monitoring system developed to gather and process data from PlanetLab nodes [27]. The server of the CoMon project collected a lot of useful information – including the temporal development of slices' 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 [4] and were conserved in the folder examples/workload/planetlab of CloudSim.

Beloglazov et al. were focusing on CPU load, and hence conserved only CPU load data. These are stored in simple text files: one file per VM per day, in which each line contains a single number, the CPU load in the given sample, as a percentage of the requested capacity. In most cases, the CPU load was measured every 5 minutes, leading to 288 measurements per day and hence 288 lines per file.

To evaluate their algorithm with realistic test data, Beloglazov et al. made the following arrangements in CloudSim, which we call together the "PlanetLab experiment":

- The `UtilizationModelPlanetLabInMemory` class in the core `org.cloudbus.cloudsim` package implements a cloudlet utilization model (see Section 3.2), which reads the utilization values from a file. At any point in time, the utilization of the cloudlet is determined based on the measurement value at that time or a linear interpolation between the two neighboring measurement values.

Table 1: Core parameters of the PlanetLab experiment, as defined by Beloglazov et al. [4]

Number of VM types:	4
VMs' number of vCPUs:	1, 1, 1, 1
VMs' requested CPU sizes [MIPS]:	2500, 2000, 1000, 500
VMs' requested RAM sizes [MB]:	870, 1740, 1740, 613
Number of PM types:	2
PMs' number of pCPUs:	2, 2
PM CPU capacities per pCPU [MIPS]:	1860, 2660
PM RAM sizes [MB]:	4096, 4096
PM power models:	HP ProLiant ML110 G4, HP ProLiant ML110 G5

Table 2: Contrasting CloudSim terminology and DISSECT-CF terminology for analogous notions

CloudSim	DISSECT-CF
Host	PhysicalMachine
Vm	VirtualMachine
Datacenter	IaaSService

- The `PlanetLabRunner` and its helper classes (`PlanetLabHelper`, `PlanetLabConstants`, and `PlanetLabRunnerConsole`) in the `org.cloudbus.cloudsim.examples.power.planetlab` package, provide a parameterizable way for setting up the experiment, including creation of a data center, setting the VM consolidation algorithm to be run every 5 minutes, creating the cloudlets using the file-based utilization model, and then starting the simulation. The parameters include the name of the input folder that contains the utilization data (each folder contains the files corresponding to one day, so choosing a folder means choosing one day), the policy for determining when a PM is to be considered overloaded, the policy for selecting which VM to migrate from an overloaded PM, and an optional additional parameter for the VM selection policy (e.g., the value of the threshold for the threshold-based policy). The number of PMs is fixed to 800.

- The `Helper` class in the `org.cloudbus.cloudsim.examples.power` package is responsible for setting up the PMs and VMs, based on the data in the `Constants` class in the same package. The most important parameter values are summarized in Table 1.

All this is readily available in CloudSim.

3.5 DISSECT-CF

DISSECT-CF stands for "DIScrete event baSed Energy Consumption simulaTor for Clouds and Federations" and is the name of an alternative cloud simulator [16]. In contrast to CloudSim, DISSECT-CF was developed by a single researcher. As a result, the code base of DISSECT-CF is more consistent and integral than that of CloudSim. The source code of DISSECT-CF is available from the project homepage[3] under the LGPL license. DISSECT-CF has been shown to deliver very accurate simulation results and at the same time be also very efficient [16]. In this work, version 0.9.6 of DISSECT-CF was used.

The fundaments of DISSECT-CF are a discrete-event time simulation (implemented by the `Timed` class) and a generic model of resource provisioning and consumption (implemented by the `hu.mta.sztaki.lpds.cloud.simulator.iaas.resou` package). On this basis, DISSECT-CF simulates the usual elements of a cloud like PMs and VMs. These notions have a one-to-one correspondence to the appropriate notions in CloudSim (see Table 2).

However, these similarities are superficial. On a more detailed level, many differences can be observed between the two simulators. Most of the differences stem from a more detailed modeling of reality in DISSECT-CF than in CloudSim. This has several important consequences:

- Creating PMs and VMs is more complicated than in CloudSim. For example, to create a PM, one must first create a `Repository` object representing the disk of the PM, connect the disk to the network, create the power model for the PM defining the power characteristics of the PM's components in the different power states (the power model is of type `EnumMap<PhysicalMachine.PowerStateKind, EnumMap<PhysicalMachine.State, PowerState>>`), then create the PM and register it in the cloud.

[3] `https://github.com/kecskemeti/dissect-cf`

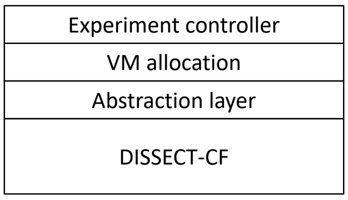

Figure 3: Layered architecture of our implementation. Upper layers use services of the lower layers.

- PMs and VMs support various states. Specifically, a PM has 4 possible states: OFF, SWITCHINGON, RUNNING, SWITCHINGOFF, whereas a VM has 11 possible states: INITIAL_TR, STARTUP, RUNNING, SUSPEND_TR, SUSPENDED, SUSPENDED_MIG, RESUME_TR, MIGRATING, SHUTDOWN, DESTROYED, NONSERVABLE. This is in sharp contrast to the very simple state modeling in CloudSim.
- Processes like starting or stopping a PM or VM, or migrating a VM between two PMs, involve transitions between multiple states. They require time and consume energy.

Some further important differences (see also Table 3 for an overview):

- In CloudSim, time is measured in seconds. In DISSECT-CF, time is measured in *ticks* and users of the simulator are free to interpret ticks the way they want. (But it is important to interpret ticks consistently across the simulation.) This also has an impact on other units: CPU capacity is measured in MIPS (million instructions per second) in CloudSim but in instructions per tick in DISSECT-CF, which again has to be consistent with the chosen interpretation of a tick.
- In CloudSim, the power consumption of a PM can be directly queried from the PM object. In DISSECT-CF, a separate `EnergyMeter` object is required which regularly monitors and aggregates the power consumption of the PM.
- In DISSECT-CF, different states of a PM may be associated with different power models. In particular, also a switched-off PM consumes energy. The power model corresponding to the current state of a PM can be queried from the PM; however, the power models corresponding to other states cannot. So it is not possible, for instance, to query from a running PM how much power it would consume if switched off (just as this is not possible in reality).
- In DISSECT-CF, the public interface of a VM does not offer a method to query the PM that is currently hosting the VM, while this can be easily done in CloudSim.
- In DISSECT-CF, there is no class representing compute tasks (like cloudlets in CloudSim). Rather, the `VirtualMachine` class offers a public method `newComputeTask` for the creation of compute tasks with given total amount of processing (number of instructions) and processing speed (instructions per tick). The created compute tasks are stored internally as instances of the class `ResourceConsumption`, but cannot be modified later on by the user code, except for cancellation. Hence, compute tasks with time-varying load are not supported directly.
- DISSECT-CF supports the notion of VM scheduler and PM scheduler. The VM scheduler is responsible for allocating newly requested VMs on appropriate PMs. The PM scheduler is responsible for switching PMs on and off. However, regular re-optimization of the VM placement (VM consolidation) is not included in DISSECT-CF[4].

It is worth mentioning that, just like CloudSim, also DISSECT-CF comes with a number of useful examples. In the case of DISSECT-CF, these are organized into a separate project called `dissect-cf-examples`.

4 Porting from CloudSim to DISSECT-CF

Since the two simulators differ in many important details, porting Beloglazov's algorithm and the PlanetLab experiment from CloudSim to DISSECT-CF involved implementing quite some code on top of DISSECT-CF. For structuring this code, we used a layered architecture, as shown in Figure 3. The *abstraction layer* provides access to the entities simulated in DISSECT-CF on a higher level of abstraction. On this basis, the *VM allocation* layer contains the code implementing the actual VM allocation algorithms. Finally, the *experiment controller* layer implements the PlanetLab experiment. In the following subsections, we describe the layers in detail.

[4]Recently, VM consolidation has been added to DISSECT-CF, but this was not available when this work was carried out.

Table 3: Some differences between CloudSim and DISSECT-CF

Aspect	CloudSim	DISSECT-CF
PM creation	1 line of code	4 lines of code
VM creation	1 line of code	3 lines of code
PM states	1 state	4 states
VM states	3 states	11 states
Turning PMs on / off	Instantaneous, no energy consumption	Time- and energy-consuming
Power consumption of turned-off PMs	0	Some given constant
Time unit	Second	Tick
CPU capacity unit	MIPS	Instructions per tick
Querying PM power consumption	Whole power model can be queried from PM	Only power model of current state can be queried; energy consumption can be measured using dedicated `EnergyMeter` object
Querying hosting PM from VM	Possible	Not possible
Representation of compute tasks	Cloudlets	No explicit representation

4.1 Abstraction layer

The API of DISSECT-CF provides a very detailed interface to the simulated entities. This allows fine-grained control and ultimately very accurate simulations. However, a disadvantage of this API is that implementing complex logic – like what we need for the VM allocation algorithm and for the PlanetLab experiment – is tedious and error-prone (see for example in Section 3.5 how complicated it is to create a PM in DISSECT-CF). In particular, the VM allocation algorithm is programmed against a much more streamlined API in CloudSim. For these reasons, we decided to first create an abstraction layer on top of DISSECT-CF that provides the same kind of high-level easy-to-use access to the necessary entities as the API of CloudSim.

The classes that make up this abstraction layer are shown in Figure 4. In the center, there are the classes `DC`, `PM`, and `VM`, providing convenient access to the underlying classes in the simulator (`IaaSService`, `PhysicalMachine`, and `VirtualMachine`, respectively). They provide the following extra functionality, compared to the underlying classes in the simulator:

- The `DC` class provides a convenient way to set up a data center. In particular, it caters for the network in the data center, a central storage, and the energy metering.
- The `PM` class provides multiple features:
  - Easy creation of PMs
  - Access to the complete power model of the PM (see below)
  - Computation of the total CPU capacity requested by, respectively allocated to, the VMs on the given PM
  - Unique identification of each PM
- The `VM` class provides the following features:
  - Easy creation of VMs
  - Possibility to specify the VM’s current CPU load as a given percentage of its capacity
  - Unique identification of each VM

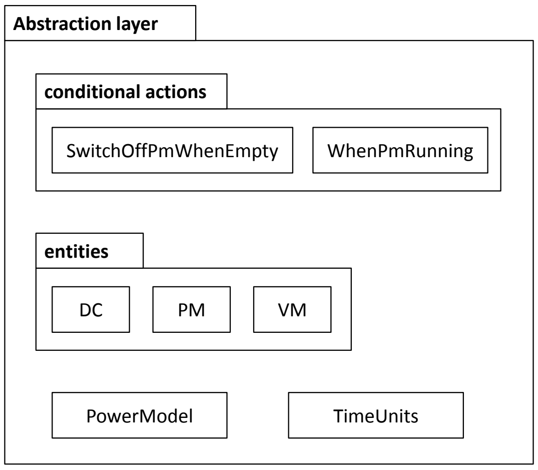

Figure 4: Contents of the abstraction layer

The `PowerModel` class provides power models for PMs, analogously to the measurements-based power models in CloudSim. That is, the measured power consumption is stored for CPU utilizations of 0%, 10%, 20%, ..., 100%, and interpolated linearly between the measured values. The power consumption of a switched-off PM is taken to be 0, as in CloudSim.

The aim of the `TimeUnits` class is to ensure consistent interpretation of ticks. As mentioned in Section 3.5, DISSECT-CF uses ticks as time unit and it is the user's responsibility to map ticks to some real-world time units, and to do it in a consistent way. Therefore, the `TimeUnits` class defines a specific value for a tick (e.g., one millisecond), and it provides static methods to convert between ticks and seconds, as well as between instructions per tick and MIPS. This way, our code can work with seconds and MIPS, using the methods of `TimeUnits` to convert them to the internal units of DISSECT-CF.

Finally, the abstraction layer provides *conditional actions* (see the upper part of Figure 4). The aim of these is to schedule future actions based on some condition. These are necessary because, as mentioned in Section 1, migrating a VM and switching on a PM take some simulated time in DISSECT-CF. This leads to different kinds of conditional actions in the context of VM allocation:

- If the VM allocator decides to migrate all VMs from a PM and shut down the PM, then the shut-down process can only be started when the migration of all VMs has completed.
- If the VM allocator decides to migrate a VM to a PM which is not yet on, then first that PM must be turned on, and the migration can start only when the PM has already reached the "running" state.
- Similarly, if the VM allocator decides to allocate a newly requested VM on a PM that is not yet on, then the PM must be first turned on, and the allocation can start only afterwards.

DISSECT-CF contains mechanisms with which these problems can be solved: by registering appropriate listeners to different kinds of system events, one can find out when a PM becomes empty or when the state of a PM becomes "running". However, the usage of listeners makes the code quite complicated.

Therefore, we created two classes that encapsulate the listener-based solutions for the above situations, hiding the complexity from the user. The `SwitchOffPmWhenEmpty` class does exactly what its name suggests: it can be used to ensure that a PM will be turned off once it becomes empty. Internally, it uses the listener-based mechanism of the simulator, but its usage is very simple: the line

```
new SwitchOffPmWhenEmpty(pm);
```

ensures that `pm` will be switched off when that becomes possible. For handling the other two situations from the above list, the `WhenPmRunning` class has been created. It is instantiated with a PM, the state of which has to be observed, and a method with its arguments, which needs to be called when the PM has reached the "running" state. Again, the class internally uses the listener-based mechanism of the simulator (and in this case also reflection), but its usage is simpler. The line

```
new WhenPmRunning(targetPm,this,"migration",vm,targetPm);
```

ensures that `vm` will be migrated to `targetPm` (by calling `this.migration(vm,targetPm)`) once `targetPm` is turned on. Similarly, the line

```
new WhenPmRunning(pm,this,"allocation",vm,pm);
```

ensures that `vm` will be allocated on `pm` (by calling `this.allocation(vm,pm)`) once `pm` is turned on.

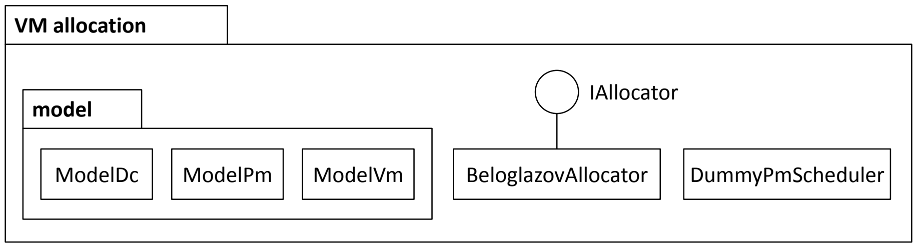

Figure 5: Contents of the VM allocation layer

4.2 VM allocation layer

The aim of the VM allocation layer is the implementation of Beloglazov's VM placement algorithm. The classes forming this layer of the architecture are depicted in Figure 5.

A challenge in implementing Beloglazov's algorithm is that some of the changes that it makes to the VM allocation are not legal, at least not in DISSECT-CF (they are legal in CloudSim, of course). In particular, when a PM is over-utilized, the algorithm removes VMs from it as long as it is over-utilized. The algorithm then looks for a new PM for each of the given VMs and places them on their new hosts; at the same time it notes that a migration needs to take place. Similarly, for each PM that is not over-utilized, the algorithm tries to free it by moving all its VMs to other PMs. For this purpose, it moves the VMs hosted by the given PM one by one, but if it is not possible to find an appropriate new host for one of them, then it undoes all moves from the given PM. Otherwise, it notes for all the VM moves from the given PM that an appropriate migration must be carried out. At the end of the algorithm, the original allocation of the VMs is restored, and then all the collected migrations are performed. This time, the migrations are "real" in the sense that they take some simulated time and the resource utilization and energy consumption of the VMs during the migration are adapted to reflect the fact that the VM is being migrated, as described in Section 3.2.

The preliminary changes that the algorithm makes during its run are possible in CloudSim, because it allows direct manipulation of the VM allocation, without using "real" migrations. However, such direct changes to the allocation of VMs without actually migrating them are not supported in DISSECT-CF (just like they are not possible in a real cloud).

To allow the algorithm to make such preliminary changes to the VM allocation in DISSECT-CF, we *decouple the model of the data center used by the algorithm from the actual data center*. This is made possible by the `model` package, which provides a model of the data center, together with its PMs and VMs (see Figure 5). The model is initialized to reflect the current PM state and VM allocation in the data center. The algorithm works on this model and can freely change it in any way, without implying any actual operation in the data center. During the run of the algorithm, the model can also become temporarily inconsistent (e.g., a VM is removed from its old host but not yet placed on a new one) without causing any problems. At the end of the algorithm, the changed model is compared to the original one to determine which PMs need to be switched on or off and which VMs need to be migrated. Then, the collected list of operations (migrations and switching on or off of PMs) is eventually carried out in the data center.

This way, the algorithm can use the same mechanism in DISSECT-CF as in CloudSim. The only difference is that in CloudSim, the simulated data center and its freely modifiable model are realized by the same set of objects. These objects can be accessed through two different APIs: free and direct manipulation during the algorithm versus the realistic cloud operations with their simulated overhead. This is a very convenient, but insecure and unrealistic design. DISSECT-CF has a stricter design, in which the simulation objects cannot be freely accessed. This is why we need the extra layer (called "model") to have a sandbox in which the algorithm can freely experiment with changes to the VM allocation, without the restrictions of the simulator.

Based on the mechanisms provided by the Abstraction layer (Section 4.1) and the `model` package, it was not difficult anymore to port the algorithm from CloudSim to DISSECT-CF. It is implemented in the `BeloglazovAllocator` class. To the outside, it exhibits the simple `IAllocator` interface with only two public methods: one for determining a new host for a VM and one for re-optimization of the allocation. The algorithm itself is implemented by a set of private methods to determine the overloaded PMs, select and remove VMs from them, migrating them to new hosts, migrating VMs from not over-loaded PMs etc. These could be easily ported from CloudSim, using the classes of the `model` package.

The last class in this layer is the `DummyPmScheduler`. As described in Section 3.5, DISSECT-CF works with a VM scheduler and PM scheduler: the VM scheduler is responsible for allocating newly requested VMs, while the PM scheduler is responsible for turning PMs on or off. The new VM allocator makes both of these unnecessary. The VM scheduler is simply bypassed by requesting the allocation of newly requested VMs from the VM allocator (through the `IAllocator` interface). However, in the case of the PM scheduler, such bypassing is not possible since the PM scheduler is automatically called by the `IaaSService` on changes to the PMs or VMs. This could be problematic since the decisions of the PM scheduler to switch a PM or or off could interfere with the decisions of the VM allocator. Therefore, we implemented our own PM scheduler, the `DummyPmScheduler`, which implements all necessary methods of a PM scheduler but does nothing. This way, the VM allocator has full control over the state of the PMs. (In an alternative design, the VM allocator could give orders to the PM scheduler about switching PMs on or off, which would be executed by the PM scheduler, and the VM allocator would not change the state of PMs directly. This might be a better fit to the current structure of DISSECT-CF, but it is also a bit more complicated.)

4.3 Experiment controller

The highest layer consists of a single class called `ExperimentController`. It is a singleton that is responsible for conducting the PlanetLab experiment using Beloglazov's algorithm by orchestrating the operation of the underlying layers.

Specifically, the tasks of the `ExperimentController` are as follows:

- Initialization:
  - Instantiation of all relevant entities (DC with PMs and VMs, `BeloglazovAllocator` etc.) and linking them with each other.
  - Reading workload data from the PlanetLab trace files.
  - Starting energy metering.
  - Creating recurring activity (see below) using the simulator's `Timed` class.
- Initial allocation of the VMs using the `BeloglazovAllocator`.
- Recurring activities according to the re-optimization interval, i.e., every 5 minutes:
  - Resizing the VMs according to the next CPU load value from the PlanetLab trace.
  - Starting a new compute task on each VM for the next re-optimization interval, corresponding to the CPU load value from the PlanetLab trace.
  - Performing re-optimization of the VM allocation using the `BeloglazovAllocator`.
- Terminating the simulation when the end of the workload trace has been reached.
- Outputting logging and debugging information, collecting and outputting statistics.

It should be noted that there is a somewhat similar experiment controller for DISSECT-CF in the `dissect-cf-examples` project, called `JobDispatchingDemo`. That class was used to perform the experiments in [16]. It reads data from a grid workload trace and creates virtual machines and compute jobs accordingly. The most important difference is that `JobDispatchingDemo` works with *jobs* that are created, have defined resource requirements, and then end, whereas our `ExperimentController` works with VMs that exist throughout the simulation but have time-varying resource requirements as defined in the trace file.

5 Insights

When porting Beloglazov's algorithm and the PlanetLab experiment from CloudSim to DISSECT-CF as described in Section 4, several interesting peculiarities of the existing implementation have been discovered. In this section, we review the most important ones.

5.1 VMs' requested resources

In Beloglazov's algorithm, both the initial VM allocation and the re-optimization of the allocation make use of the `findHostForVm` method, which is used to find a new PM for a given VM. This method iterates over all available PMs and checks whether they have sufficient resources to host the VM. Finally, from the PMs that have sufficient resources, the one with the best energy efficiency is chosen.

To check whether a given PM has sufficient resources to host a given VM, the method looks at the *requested* resources (CPU and RAM) of the VM. The handling of requested resources of a VM is somewhat peculiar in

CloudSim: when the VMs are first created at the beginning of the experiment, the requested resources of the VM are those nominal values that are given in Table 1. Thus these values are used when the VMs are initially allocated to PMs. Starting with the first re-optimization though, the CPU values are modified according to the PlanetLab trace, leading to significantly lower CPU loads. Moreover, the RAM utilization values of the VMs are set to zero. This means that the VMs occupy much less CPU and RAM from the first re-optimization onwards than when they are initially placed.

As a specific example, using the PlanetLab trace from 2011-03-09 and a utilization threshold of $\tau = 0.8$, the initial allocation leads to 751 switched-on PMs. However, after the first re-optimization, this number drops to 316.

It should be noted that, although the requested RAM of the VMs is set to 0 at the first re-optimization, this does not influence the selection of the VM to be migrated from an over-utilized PM: this decision is still based on the RAM values as indicated in Table 1.

5.2 PMs' energy efficiency

As already mentioned, the `findHostForVm` method in Beloglazov's algorithm chooses for the given VM the PM with the best energy efficiency among the ones with sufficient capacity. More precisely, the PM is chosen where allocating the new VM would lead to the *lowest increase in power consumption*.

In the PlanetLab experiment, there are two kinds of PMs, simulating HP ProLiant servers of the 4th and 5th generation, respectively. The corresponding power consumption characteristics are shown in Figure 6. As can be seen, both are roughly linear and the curve corresponding to the G5 server is higher. Hence, looking at this picture, the intuition is that – all else being equal – G4 servers are preferable to G5 servers and the current load level of the server does not matter for the choice of the most power-efficient PM.

However, the algorithm bases its decision on the increase in power consumption that a given VM would cause, which is proportional to the *slope* of the power consumption characteristic curve at the current load level of the PM. Therefore, Table 4 shows the slope of the power consumption characteristics of the used PM types for different CPU load intervals. Lower values are preferred by the algorithm. The table shows that the – perfectly plausible – preference of the algorithm for lower increases in power consumption actually leads to strange and unintuitive bias. For example, the slope of the power efficiency curve of G5 servers is more favorable between 0 and 30% than between 30 and 60%. This bias (which we believe was not intentional) may lead to suboptimal decisions of the algorithm. For example, when the algorithm has to choose from two G5 servers, one of them with CPU load 10% and the other with CPU load 50%, for a VM whose CPU load corresponds to 10% of the PMs' capacity, the algorithm will choose the PM with CPU load of 10% because of its perceived better power efficiency. However, this is against the principle of VM consolidation, which would call for packing as much load on the already well utilized PM as possible and trying to empty the lightly loaded PM.

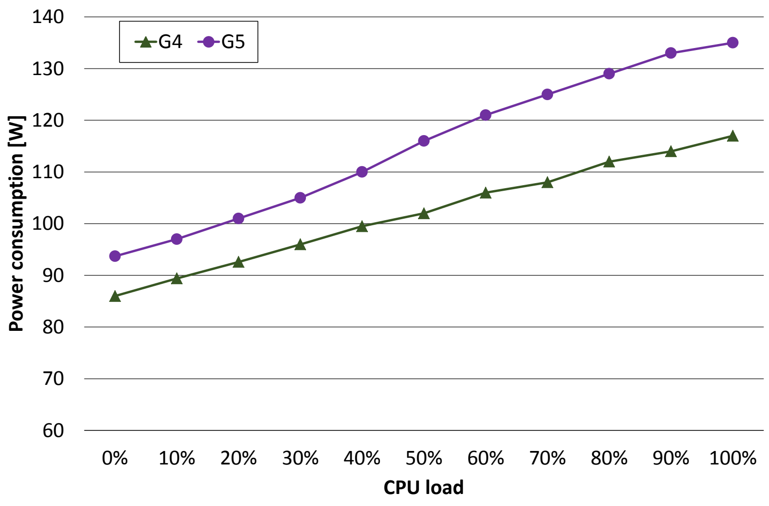

Figure 6: Power consumption characteristics of the used PM types

5.3 Migration map

As explained in Section 4.2, Beloglazov's algorithm first makes preliminary changes to the VM placement without any "real" migrations, while the migrations that should be really carried out are collected. At the end of the algorithm, the collected migrations are actually carried out.

Table 4: Slope of the power consumption characteristic of the used PM types for different CPU load intervals

PM type	0-10%	10-20%	20-30%	30-40%	40-50%	50-60%	60-70%	70-80%	80-90%	90-100%
G4	0.34	0.32	0.34	0.35	0.25	0.4	0.2	0.4	0.2	0.3
G5	0.33	0.4	0.4	0.5	0.6	0.5	0.4	0.4	0.4	0.2

A small but important implementation detail is how the collected migrations are stored. In CloudSim, this is done using a `List<Map<String,Object>>` data structure. Each element of this list encodes one particular migration as a map with two elements: `"vm" -> vm`, `"host" -> targetPm`, where `vm` is the object representing the VM to be migrated and `targetPm` is the object representing the PM where the VM should be migrated to.

We believe this data structure is a suboptimal choice. Instead, we propose to use a `Map<VM,PM>`, where each map entry consists of a VM to be migrated and the corresponding target PM. This more direct way of storing the necessary information is associated with significantly less memory overhead and makes it possible to quickly and easily find out where a VM is to be migrated. Even more importantly, it ensures that only one migration will be stored for each VM. With Beloglazov's algorithm, it is possible that a VM v is first moved from PM p_1 to p_2 and later from p_2 to p_3. Using the implementation in CloudSim, both migrations are recorded and then carried out. With the proposed implementation, first the pair (v, p_2) is inserted into the migration map, which is later overwritten by (v, p_3). As a result, only one migration will be carried out: v will be migrated directly to p_3. Thus, an unnecessary migration is skipped.

5.4 Energy metering

In CloudSim, energy consumption during the PlanetLab experiment is measured as follows:

- Every five minutes, the CPU load of each PM is determined based on the CPU load of the hosted VMs.
- Based on the CPU load and the power consumption characteristic of the PM, its current power consumption is determined.
- The energy consumption ΔE of the PM in the time window $[t-\Delta t, t)$ is calculated using linear interpolation between the power consumption values at time $t - \Delta t$ and t:

$$\Delta E = \begin{cases} \frac{W_1+W_2}{2} \cdot \Delta t, & \text{if } W_1 > 0 \\ 0, & \text{if } W_1 = 0 \end{cases} \tag{1}$$

  where W_1 is the power consumption at time $t - \Delta t$ and W_2 is the power consumption at time t, and Δt is 5 min.
- The energy consumption of the PMs is aggregated to obtain the energy consumption of the whole DC.

In DISSECT-CF, the realization of energy metering is different. There is no easy way to query the current power consumption of a PM. Instead, each PM may have an associated energy meter object that regularly measures its energy consumption. The energy consumption of the PM between two measurement points is calculated based on the amount of CPU processing that has taken place during this interval, which is proportional to the average power consumption. In contrast to CloudSim, the frequency of power measurements can be configured independently from the re-optimization interval (it could be even set differently for each energy meter object). The choice of power measurement frequency constitutes an important trade-off, because higher measurement frequencies result in higher precision but also higher execution time for the simulator (see Section 6.2 for more details).

The following differences between energy measurement in the two simulators may lead to different measurement results:

- In CloudSim, the first measurement of power consumption is done at time 0, when the CPU load of the VMs is 0. As a result, the power consumption of each PM is assumed to be 0 at this point. As a consequence of the second case in Equation (1), the energy consumption of the DC in the first 5 minutes is erroneously taken to be 0. In DISSECT-CF, energy consumption is correctly estimated for the first period as well, leading to higher energy consumption values in DISSECT-CF for the first five minutes.
- Power consumption is linearly interpolated between measurement points in CloudSim, whereas this is not done in DISSECT-CF (i.e., the last measured CPU load and hence power consumption remains unchanged until the next measurement). As illustrated in Figure 7, energy consumption will be higher in CloudSim if

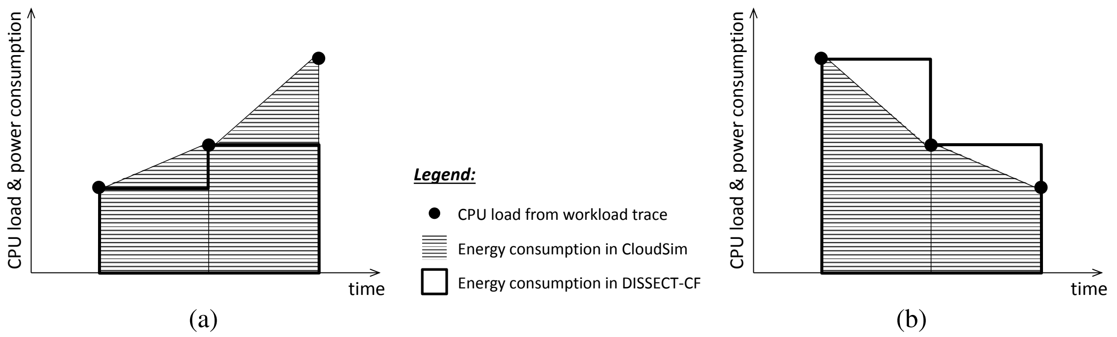

Figure 7: Energy metering in the two simulators for (a) increasing workload and (b) decreasing workload. For simplicity it is assumed that the energy metering points in DISSECT-CF coincide with the points in time when new workload data become available.

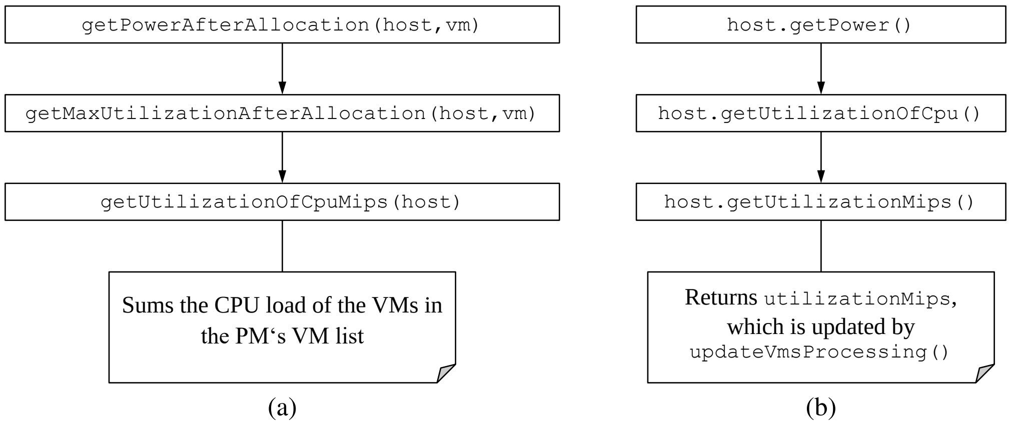
getPowerAfterAllocation(host,vm)
getMaxUtilizationAfterAllocation(host,vm)
getUtilizationOfCpuMips(host)
Sums the CPU load of the VMs in the PM‘s VM list
(a)
host.getPower()
host.getUtilizationOfCpu()
host.getUtilizationMips()
Returns utilizationMips, which is updated by updateVmsProcessing()
(b)

Figure 8: Chains of method calls for computing power consumption (a) after and (b) before a new VM is allocated to the PM

the load increases, whereas it will be higher in DISSECT-CF if the load decreases. It is not possible to tell which the "correct" result is because the real development of the load is unknown between the measurement points and the two implementations take different assumptions on it. If a workload contains similar amounts of load increases and decreases over time, then the two effects neutralize each other, leading to no systematic bias between the results of the two simulators.

- The energy consumption results of the two simulators differ also because of the already mentioned differences in states and state transitions. These include the possibly non-zero power consumption of switched-off PMs, the energy consumption of the switch-on and switch-off processes as well as that of migration processes. Since CloudSim neglects these, whereas DISSECT-CF takes them into account, this leads to higher energy consumption values in DISSECT-CF.

As a result of these three effects and assuming that the workload contains similar amounts of increases and decreases over time, DISSECT-CF will report higher energy consumption values for the same scenario than CloudSim.

5.5 Computing the increase in power consumption

A core idea of the algorithm of Beloglazov et al. is to select for hosting VM v the PM that has enough resources for hosting v and where v would lead to the lowest increase in power consumption. Therefore, the algorithm computes for each PM candidate the increase in power consumption that v would cause, as follows (in the method `findHostForVm` of `PowerVmAllocationPolicyMigrationAbstract`):

```
powerDiff=getPowerAfterAllocation(host,vm)-host.getPower()
```

Interestingly, the power consumption of the PM before and after allocating the VM to it are computed in two different ways. As shown in Figure 8(a), the power consumption after allocation is computed using the allocator's own `getPowerAfterAllocation` method, which uses other methods of the allocator to figure out the CPU load of the PM after allocation of the new VM. In the end, it is the allocator's `getUtilizationOfCpuMips` method that does this by summing the CPU load of the VMs that are in the PM's VM list.

As Figure 8(b) shows, the computation of the PM's current power consumption is calculated differently: not by the allocator, but by using the simulator's standard methods for that purpose. In particular, the CPU utilization of the PM is determined based on the PM's `utilizationMips` attribute. The value of this attribute is set by the `updateVmsProcessing` method after changes were made to the PM's allocation to equal the sum of the CPU loads of the VMs hosted by the PM.

In principle, the two ways of computing power consumption should lead to consistent results. But this is often not the case. This is a consequence of the algorithm's transient moves of VMs among PMs without real migrations. When the algorithm moves a VM this way from PM p_1 to PM p_2, it is removed from the VM list of p_1 and added to the VM list of p_2, leading to appropriate changes in the power consumption reported according to Figure 8(a). However, the `updateVmsProcessing` method is not run in this case (it will be run only later when the VM is properly migrated from p_1 to p_2), so that the power consumption reported according to Figure 8(b) remains unchanged. Therefore, `powerDiff` values computed for p_1 and p_2 will be wrong afterwards.

This means that at the beginning of the algorithm, the `powerDiff` values it computes are accurate, but as time goes by, it happens more and more often that the `powerDiff` values are inaccurate, thus impacting the algorithm's behavior. The impact on the algorithm's behavior can be understood by continuing the above example in which a VM v has been moved (without a real migration) from p_1 to p_2. The next time the algorithm looks for a host for a new VM, when it evaluates p_1, the result will be an unrealistically low `powerDiff` value. This is because, when computing the `powerDiff` for p_1, the contribution of v is not included in `getPowerAfterAllocation(host,vm)`, but it is included in `host.getPower()`. When evaluating p_2, the opposite happens: the contribution of v is included in `getPowerAfterAllocation(host,vm)`, but it is not included in `host.getPower()`, leading to an unrealistically high `powerDiff` value. Since the algorithm prefers PMs with low `powerDiff`, the consequence is that the algorithm prefers p_1 for hosting the new VM over any untouched PM, and it prefers the untouched PMs over p_2.

We assume this behavior was not intended by the inventors of the algorithm. In particular, it works against the idea of consolidation. In the above example, the goal of consolidation would be to empty p_1 and to pack as much load as possible on p_2, but the inconsistent computation of `powerDiff` values leads to a bias in the opposite direction.

5.6 Notes on our implementation in DISSECT-CF

For the behavior reported in Sections 5.1 and 5.2, we decided to reproduce it in our implementation in DISSECT-CF. Our main reason for that was to foster comparability of the two implementations. Moreover, in the case of the behavior described in Section 5.1 (the drop of the considered VM resources after initial allocation), there can be a reason for that: e.g., the provider initially does not know the exact resource requirements of the VMs, but figures it out through monitoring when they are already running. Regarding the behavior described in Section 5.2 (the dependence on the local slope of the power consumption characteristic), we felt this was a part of the algorithm even though its unintended effects have become clear.

In contrast, we did not reproduce the behavior described in Sections 5.3 and 5.5 in our implementation in DISSECT-CF. In the case of Section 5.3 (data structure for migrations allowing multiple migrations of the same VM) we felt it is not an intrinsic property of the algorithm, just a suboptimal implementation detail. Regarding Section 5.5 (inconsistent prediction of increases in power consumption), we would go even one step further and consider it a bug in the existing implementation of the algorithm.

The behavior described in Section 5.4 is not subject to our implementation decisions, but given properties of the two simulators.

Finally, it should be noted that the implementation of the algorithm in CloudSim remained unchanged.

6 Experimental results

This section reports the results of running the PlanetLab experiment with Beloglazov's algorithm in the two simulators. All measurements were performed on a Lenovo ThinkPad X1 laptop with Intel Core i5-4210U CPU running at 1.70GHz and 8GB RAM.

6.1 Comparing the results achieved with the two simulators

Test cases can be determined by using different days of the PlanetLab trace and different values for the over-utilization threshold τ. In a first set of experiments, we fixed $\tau = 0.9$ and compared the implementations of the algorithm in the two simulators for different days of the PlanetLab trace. The comparison results are reported for

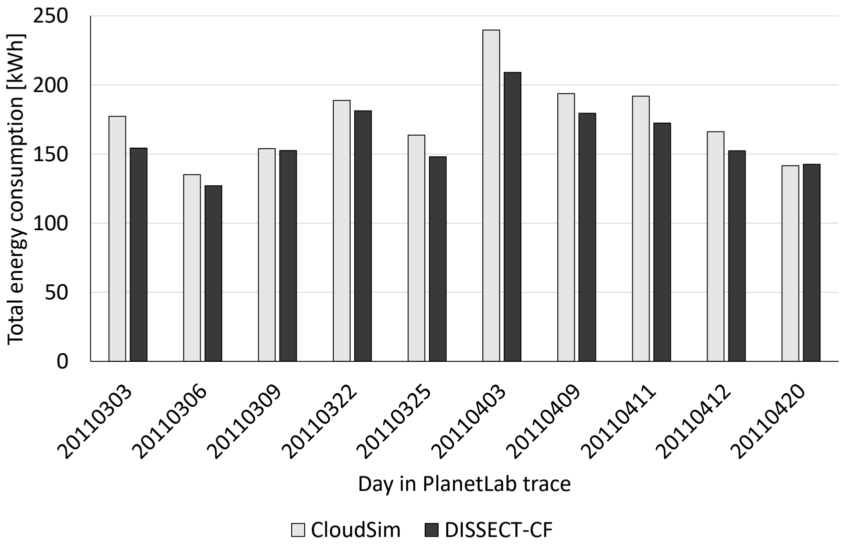

Figure 9: Energy consumption in the two simulators

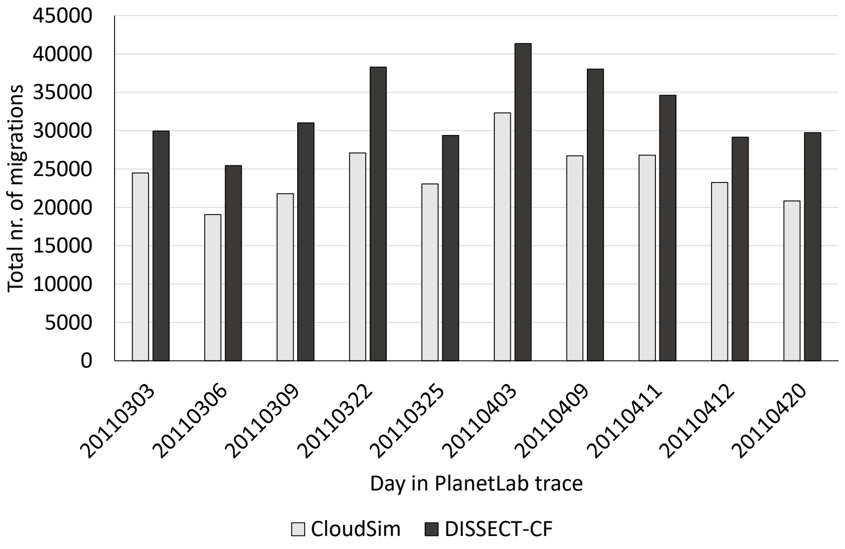

Figure 10: Number of migrations in the two simulators

three metrics: total energy consumption (Figure 9), total number of migrations (Figure 10) and total execution time of the simulator (Figure 11).

In terms of the total energy consumption, the implementation in DISSECT-CF leads to an average reduction of 7%. With the exception of a single outlier (the last day, for which the implementation in DISSECT-CF leads to 1% increase in energy consumption), the implementation in DISSECT-CF always leads to a reduction, which is in the range of 1% to 13%. This better energy consumption comes at the cost of a higher number of migrations – increased by 22% to 43%, with an average of 33%.

These systematic differences between the results of the two implementations of the algorithm can be understood in light of the findings reported in Section 5. In particular, the issues shown in Section 5.5 explain why the implementation in CloudSim is not as effective in consolidating the VMs as it could be, leading to fewer migrations and higher energy consumption. The insights from Section 5.4 explain why the differences in energy consumption are not even higher.

In terms of execution time, we can observe a consistent advantage of DISSECT-CF over CloudSim. The reduction in execution time ranges from 82% to 87% with an average of 84%. This is all the more remarkable since, as we saw previously, the DISSECT-CF implementation of the algorithm leads to more migrations than its CloudSim counterpart. That is, DISSECT-CF has to simulate more migrations, yet needs much less time for it.

We also performed experiments with other values of τ. However, the results followed an analogous pattern and are hence omitted here.

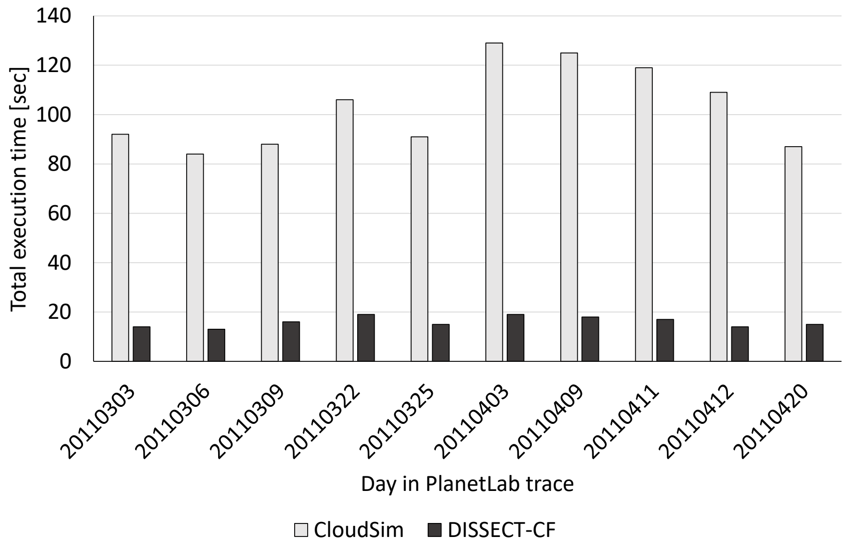

Figure 11: Execution time of the two simulators

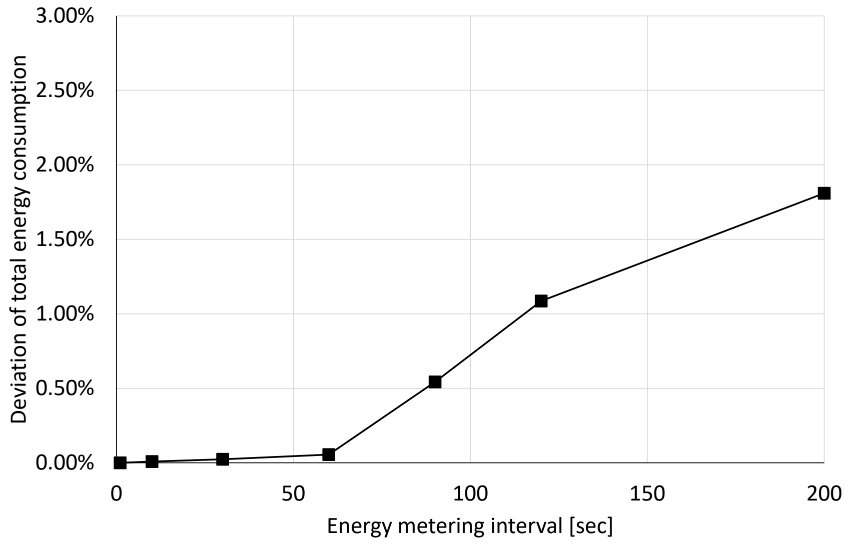

Figure 12: Deviation of the measured total energy consumption from the most accurate value, as a function of the energy metering interval

6.2 The effect of the energy metering interval in DISSECT-CF

In the case of DISSECT-CF, a further important parameter is the energy metering interval. It is the time between consecutive power consumption measurements of the energy meters associated with the PMs within the simulator. Higher frequency of measurements leads to improved accuracy but also to increased overhead.

In the experiments reported in Section 6.1, the energy metering interval was fixed to 60 seconds. Now we evaluate the effect of different values of this parameter on the obtained energy consumption and on the execution time of the simulator. (The number of migrations is not affected by this parameter.) For this experiment, we used the PlanetLab data from 2011-03-06 and a PM overload threshold of $\tau = 0.9$.

Figure 12 shows the effect on the measured total energy consumption, in terms of the relative deviation from the most accurate value (measured with an energy metering interval of 1 second). As can be seen, as long as the energy metering interval is below 100 sec, the inaccuracy of the energy metering remains below 1%. Even for an energy metering interval of 200 sec, the inaccuracy is below 2%.

Figure 13 shows the effect of the energy metering interval on the total execution time of the simulator. As can be seen, if the energy metering interval is very small, energy metering dominates the execution time of the simulator: reducing the energy metering interval from 10 sec to 1 sec increases execution time by a factor of 7. However, for energy metering intervals higher than 10 sec, the contribution of energy metering to the overall execution time of the simulator diminishes, and so does the effect of the energy metering interval on the total

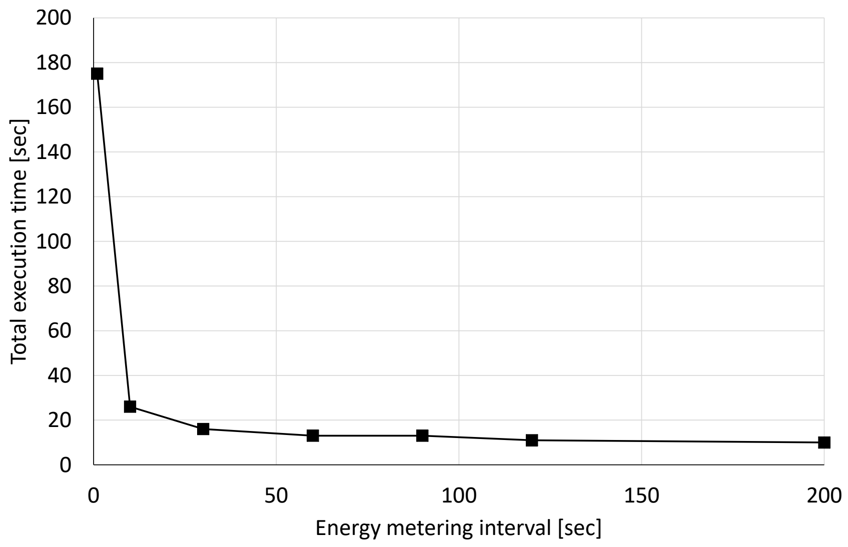

Figure 13: Total execution time of the simulation, as a function of the energy metering interval

execution time. For example, increasing the energy metering interval from 60 sec to 200 sec reduces execution time only by 23%. Similar results were reported in [16].

Thus it can be concluded that there is a wide range of values for the energy metering interval – e.g., from 30 sec to 120 sec – that leads to both very low overall inaccuracy and to low execution time.

7 Discussion

There are several lessons learned from our experience with porting a VM placement algorithm and its evaluation environment from one cloud simulator to another.

The biggest problem was the very tight integration between the implementation of the algorithm and the general-purpose simulation facilities of CloudSim. This made it difficult to port the algorithm to another simulator, suggesting that it would also be difficult to create a real-world implementation of the algorithm. Moreover, this mixture of concerns made it hard to verify the correctness of the implementation of the algorithm in the first place. The behavior described in Section 5.5 is a prime example showing how the lack of clearly cut responsibilities led to an unwanted behavior.

Instead, we suggest to loosely couple the VM placement algorithm to the rest of the simulator, using a well-defined interface along the lines of Figure 1. It is important to note that a clear interface does not only mean a well-defined entry point. Also the existing CloudSim implementation of the algorithm had a well-defined entry point with well-defined inputs and outputs. The problem was that the algorithm used many other interfaces to the rest of the simulator for querying and directly manipulating it, beyond the ones exposed at the defined entry point.

DISSECT-CF did not contain any VM placement re-optimization algorithm, thus when we started implementing Beloglazov's algorithm in DISSECT-CF, a loosely coupled approach was the logical choice, naturally leading to a clear separation of concerns. The biggest challenge was to bridge the gap between the detailed technical API offered by DISSECT-CF and the higher level of abstraction on which the algorithm was specified. The abstraction layer presented in Section 4.1 proved to be very useful for this purpose.

A closely related issue is that the algorithm must be able to make tentative changes to the placement of VMs and the state of PMs, without any consequences on the simulated system. The existing CloudSim implementation offers ad hoc solutions for this, which proved to be error-prone. In our experience, our approach of having an explicit abstract model of the PMs and VMs as a sandbox for the algorithm leads to better separation of concerns and hence to cleaner code. This also helps in defining a clear interface between the algorithm and the rest of the simulator.

Our suggested approach introduces an overhead because each time the algorithm is run, it has to first extract its model of the current system state. However, as shown by the experimental results of Section 6, this overhead is not too high since the DISSECT-CF implementation is significantly faster even with this overhead.

So far we focused on the interface between the algorithm and the simulator. However, there is a similar challenge regarding the interface between the logic of the PlanetLab experiment and the rest of the simulator. In CloudSim, these concerns were also somewhat intertwined. This may be one of the reasons behind the strange effect that the energy consumption of the first five minutes of the PlanetLab experiment is taken to be zero (see Section 5.4). The layered architecture that we proposed in Figure 3 provides clear separation of concerns, leading to code the correctness of which is easier to assure.

As explained in Section 5 and empirically demonstrated in Section 6, the implementation of Beloglazov's algorithm in CloudSim is not as effective in consolidating VMs as it could be. This is in line with the empirical findings of previous research that indicated that this algorithm – more precisely, its implementation in CloudSim – compares unfavorably to some similar competitors [24]. It also opens the question whether an improved implementation of the same algorithm would lead to better results.

In terms of the execution time of the simulation, our experiments confirmed that DISSECT-CF has a clear advantage over CloudSim, which is important for researchers wanting to execute many simulation runs to compare different parameter configurations of their algorithm. We also found that DISSECT-CF has excellent documentation. The only downside was the relatively low abstraction level of its API; however, this could be mitigated with the suggested abstraction layer of Section 4.1, without losing the speed advantage.

Finally, it should be noted that there are many aspects that a VM placement algorithm may take into account but which are not considered by the algorithm of Beloglazov et al. and hence were also not considered in this paper. Examples include the aspects related to data transfer among VMs (such as minimizing latency, minimizing bandwidth usage, considerations of network topology, competition between data transfer among VMs and VM migrations, energy consumption of network devices), more sophisticated service level agreements, VM priorities, affinity constraints, or thermal aspects. Further research is needed to better understand the extent to which the choice of simulator affects the ability to take these aspects into account in VM placement.

8 Conclusions

This paper described our experience with porting a VM placement algorithm and its evaluation setup from one cloud simulator (CloudSim) to another (DISSECT-CF). Our main conclusions can be summarized as follows:

- Although the two simulators simulate the same real-world phenomena and offer similar simulation constructs, subtle differences (e.g., states of PMs and VMs, timing and synchronicity behavior) can make it challenging to port the same algorithm from one simulator to the other.
- Mixing the code of the algorithm or that of the experiment control logic with the rest of the simulator leads to error-prone code. Instead, we propose a layered architecture with clear separation of concerns and well-defined interfaces.
- There was a gap in the level of abstraction between the algorithm's existing implementation and the API offered by DISSECT-CF, but this could be bridged with an appropriate abstraction layer.
- The algorithm needs the possibility to freely move VMs between PMs without a direct impact on the simulated system. This can be achieved with a decoupled model of the simulated system.
- Our empirical results reinforced that DISSECT-CF has a considerable advantage in execution time over CloudSim, even with the overhead added by the abstraction layer and the decoupled model of the simulated system.
- As a side effect, we found some probably unwanted phenomena relating to the implementation of the algorithm in CloudSim which explain why it is not as effective in consolidating VMs as it could be.

Thus, our study reveals for the first time how important it is to choose the right simulator and to implement a VM placement algorithm in an appropriate way in the simulator. The gained insights will hopefully contribute to the correct implementation and accurate and efficient evaluation of VM placement algorithms.

Acknowledgement

This work was partially supported by the European Union's Horizon 2020 research and innovation programme under grant 731678 (RestAssured) and the Hungarian Scientific Research Fund (Grant Nr. OTKA 108947). Useful discussions with Dr. Gábor Kecskeméti are gratefully acknowledged.

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] Mansoor Alicherry and TV Lakshman. Network aware resource allocation in distributed clouds. In *Proceedings of IEEE Infocom*, pages 963–971, 2012.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Digital Power Group. *The cloud begins with coal – Big data, big networks, big infrastructure, and big power*. 2013.

[10] Eugen Feller, Louis Rilling, and Christine Morin. Snooze: A scalable and autonomic virtual machine management framework for private clouds. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 482–489, 2012.

[11] 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.

[12] Tom Guérout, Thierry Monteil, Georges Da Costa, Rodrigo Neves Calheiros, Rajkumar Buyya, and Mihai Alexandru. Energy-aware simulation with DVFS. *Simulation Modelling Practice and Theory*, 39:76–91, 2013.

[13] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. iFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.

[14] Takahiro Hirofuchi, Adrien Lebre, and Laurent Pouilloux. SimGrid VM: Virtual machine support for a simulation framework of distributed systems. *IEEE Transactions on Cloud Computing*, 6(1):221–234, 2018.

[15] Jun-Kwon Jung, Nam-Uk Kim, Sung-Min Jung, and Tai-Myoung Chung. Improved CloudSim for simulating QoS-based cloud services. In Youn-Hee Han, Doo-Soon Park, Weijia Jia, and Sang-Soo Yeo, editors, *Ubiquitous Information Technologies and Applications*, volume 214 of *Lecture Notes in Electrical Engineering*, pages 537–545. Springer, 2013.

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

[17] 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.

[18] 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.

[19] Md Mahmud, Mahbuba Afrin, Md Razzaque, Mohammad Mehedi Hassan, Abdulhameed Alelaiwi, Majed Alrubaian, et al. Maximizing quality of experience through context-aware mobile application scheduling in cloudlet infrastructure. *Software: Practice and Experience*, 46(11):1525–1545, 2016.

[20] 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.

[21] Z. Á. Mann. Rigorous results on the effectiveness of some heuristics for the consolidation of virtual machines in a cloud data center. *Future Generation Computer Systems*, 51:1–6, 2015.

[22] 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.

[23] Zoltán Ádám Mann. Modeling the virtual machine allocation problem. In *Proceedings of the International Conference on Mathematical Methods, Mathematical Models and Simulation in Science and Engineering*, pages 102–106, 2015.

[24] Zoltán Ádám Mann and Máté Szabó. Which is the best algorithm for virtual machine placement optimization? *Concurrency and Computation: Practice and Experience*, 29(10):article e4083, 2017.

[25] Alberto Núñez, Jose L Vázquez-Poletti, Agustin C Caminero, Gabriel G Castañé, Jesus Carretero, and Ignacio M Llorente. iCanCloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, 10(1):185–209, 2012.

[26] Hamza Ouarnoughi, Jalil Boukhobza, Frank Singhoff, Stéphane Rubini, and Erwann Kassis. Considering I/O processing in CloudSim for performance and energy evaluation. In *International Conference on High Performance Computing*, pages 591–603, 2016.

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

[28] 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.

[29] 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*, 47(4):505–521, 2017.

[30] Nguyen Quang-Hung, Pham Dac Nien, Nguyen Hoai Nam, Nguyen Huynh Tuong, and Nam Thoai. A genetic algorithm for power-aware virtual machine allocation in private cloud. In *Information and Communication Technology – EurAsia Conference*, pages 183–191, 2013.

[31] RightScale. Cloud computing trends: 2017 state of the cloud survey. `http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2017-state-cloud-survey`, 2017.

[32] Georgia Sakellari and George Loukas. A survey of mathematical models, simulation approaches and testbeds used for research in cloud computing. *Simulation Modelling Practice and Theory*, 39:92–103, 2013.

[33] Brett Snyder, Robert C Green, Vijay Devabhaktuni, and Mansoor Alam. ReliaCloud-NS: A scalable web-based simulation platform for evaluating the reliability of cloud computing systems. *Software: Practice and Experience*, 48(3):665–680, 2018.

[34] 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.

[35] 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.

[36] Michael Tighe, Gaston Keller, Michael Bauer, and Hanan Lutfiyya. DCSim: A data centre simulation tool for evaluating dynamic virtualized resource management. In *8th International Conference on Network and Service Management*, pages 385–392, 2012.

[37] Pedro Velho, Lucas Mello Schnorr, Henri Casanova, and Arnaud Legrand. On the validity of flow-level TCP network models for grid and cloud simulations. *ACM Transactions on Modeling and Computer Simulation*, 23(4), 2013. Article 23.