J Grid Computing (2015) 13:375–389
DOI 10.1007/s10723-014-9312-9

Towards a Greener Cloud Infrastructure Management using Optimized Placement Policies

J. A. Pascual · T. Lorido-Botrán · J. Miguel-Alonso · J. A. Lozano

Received: 23 June 2014 / Accepted: 12 September 2014 / Published online: 26 September 2014

Abstract Cloud infrastructures are designed to simultaneously service many, diverse applications that consist of collections of Virtual Machines (VMs). The placement policy used to map applications onto physical servers has important effects in terms of application performance and resource efficiency. We propose enhancing placement policies with network-aware optimizations, trying to simultaneously improve application performance, resource efficiency and power efficiency. The per-application placement decision is formulated as a bi-objective optimization problem (minimizing communication cost and the number of physical servers on which an application runs) whose solution is searched using evolutionary techniques. We have tested three multi-objective optimization algorithms with problem-specific crossover and mutation operators. Simulation-based experiments demonstrate how, in comparison with classic placement techniques, a low-cost optimization results in improved assignments of resources, making applications run faster and reducing the energy consumed by the data center. This is beneficial for both cloud clients and cloud providers.

Keywords Cloud computing · VM placement · Multi-objective optimization · Energy consumption · Tree-network data center topology

J. A. Pascual · T. Lorido-Botrán (✉) · J. Miguel-Alonso
Intelligent Systems Group, Department of Computer Architecture and Technology, University of the Basque Country, UPV/EHU, Paseo Manuel de Lardizabal, 1, 20018, Donostia-San Sebastián, Spain
e-mail: tania.lorido@ehu.es

J. A. Pascual
e-mail: joseantonio.pascual@ehu.es

J. Miguel-Alonso
e-mail: j.miguel@ehu.es

J. A. Lozano
Intelligent Systems Group, Department of Computer Science and Artificial Intelligence, University of the Basque Country, UPV/EHU, Paseo Manuel de Lardizabal, 1, 20018, Donostia-San Sebastián, Spain
e-mail: ja.lozano@ehu.es

1 Introduction

In recent years, the utilization of cloud infrastructures to host applications has widely spread. The characteristic that makes these cloud systems so appealing is their elasticity, that is, resources can be acquired on demand, depending on the needs of the time-varying application, but paying only for those actually booked (a scheme known as pay-as-you-go). Virtualization technologies enable the cloud infrastructure to provide such elastic usage. The resources offered by physical servers, organized in several data centers, are provided in the form of abstract compute units that are implemented as Virtual Machines (VMs). Each VM is assigned a pre-configured set of resources, including:

number of cores, amount of memory, disk and network bandwidth.

Virtualized data centers support a large variety of applications, including batch jobs (typically used for scientific applications), and web applications (e.g. an online bookshop). Each application is deployed on a set of VMs, which can be allocated to any collection of physical servers in the data center. The problem of assigning a physical location to each VM is known as *VM placement* and it is performed by the manager of the cloud infrastructure. This manager is typically called the Infrastructure-as-a-Service (IaaS) provider.

The challenge for the provider is to host a large and diverse set of applications (VM sets from different clients) in the infrastructure trying to (1) maximize its revenue and (2) provide a good service to the clients. An adequate application placement would be able to maximize the resource usage of physical servers and reduce the energy consumption of the data center, for example, by turning off (or setting to idle state) the inactive servers and network elements (typically, switches). At the same time, the infrastructure management policies should trade off the obtained revenue with the Quality of Service (QoS) agreed with the client, guaranteeing that each application receives the resources paid for.

The VM placement problem has been extensively explored in the literature (e.g. [10, 12, 17, 23]). Most efforts have been directed towards optimizing the usage of CPU, memory and disk resources, and reducing the energy consumption of physical servers. However, not enough attention has been paid to the utilization of the network. An inappropriate placement of VMs with heavy communication requirements could lead to the saturation of certain network links, with the subsequent negative impact on applications (longer execution or response times). Besides, as stated in [15], the network power has been estimated at 10–20 % of the overall power consumption. For this reason, the VM placement policy should try to reduce not only the use of physical servers, but also the use of network links and switches to reduce the total power footprint.

The most common topology of data center networks is a tree of switches arranged in several tiers. The communication latency of any pair of VMs depends on the distance between the physical servers in which they are allocated. This, in turn, depends on their position in the tree. Distance is measured as the number of hops from the sender VM to the recipient. The collection of VMs forming an application communicate among them following a certain communication pattern. In web applications, the VMs are arranged into several layers and there may be intra and inter-layer communication. Other patterns are possible, depending on the particular characteristics of the application.

Based on the communication pattern of an application, and with an estimation of the workload imposed by its end-users, it is possible to approximate the input/output network bandwidth needed by each VM. The most communicative VM subsets should be placed as *close* as possible (minimizing the distance between them in terms of network hops). This means using the minimum number of physical servers, because intra-server communication is the cheapest. The constraint is that the external aggregated bandwidth required by all the VMs in a server, from the same or from different applications, cannot exceed the bandwidth of its network connection.

Two examples of common VM placement policies that are used in data centers are first fit (FF) and round robin (RR). Each of them has a different characteristic, that the infrastructure manager has to consider to choose the "right" one. FF simply selects the required physical resources consecutively, starting from the first available server. The use of this policy results in a higher utilization of the active servers because it tends to fill the possible gaps inside them. For the same reason, the number of active servers is expected to be reduced, thus saving energy. RR tries to equalize the utilization of all servers to avoid excessive wearing-out of server subsets and thermal peaks. The chosen policy affects not only the use of the infrastructure, but also the applications running on it. In the long term, FF tends to place applications in several, non consecutive nodes ("consecutive" has to be read in terms of network position), whereas RR favors contiguous placement of all the VMs of an application. This has an impact in the use of network elements (and the network-related power used) and on the performance of the applications.

In this work we demonstrate how it is possible to take these policies as starting points and use evolutionary optimization techniques to find placements that improve the benefits for both the infrastructure provider and the application. In particular, we evaluate three well-known multi-objective evo-

lutionary algorithms with problem-specific crossover and mutation operators. They implement mechanisms to converge rapidly to high quality placements. Experiments demonstrate that allocating applications using optimization-based policies results in a lower utilization of resources (servers, networking elements) while improving the performance of applications.

The remaining of this paper is organized as follows. After a review of the literature (Section 2), in Section 3 we provide models for cloud applications, input workloads, data center organizations, and the energy consumed by servers and switches. Then, we formulate VM placement as a multi-objective optimization problem and describe the evolutionary techniques used (Sections 4 and 5). We assess the benefits of our approach using the experiments defined in Section 6, whose results are discussed in Section 7. We conclude in Section 8 with some conclusions and future lines of work.

2 Related Work

This work is focused on the problem of performing the initial placement of the collection of VMs that constitute a cloud application. We argue that making a *suitable* initial decision about VM placement is essential to keep the data center in a near-optimal state, both in terms of energy consumption and resource utilization, and also to reduce the future need of consolidation. Consequently, fewer VM migrations will be required, that directly implies less network overload due to transfers. This review of the literature pays special attention to papers that target the initial placement of applications.

Open-source tools for cloud management use rather simple placement policies. For example, Eucalyptus [2] implements FF and RR strategies that only consider the VM requirements and the availability of resources. It also implements a PowerSave policy that is similar to the ranking algorithm available in OpenNebula [5]: choosing first the most used servers (with room for the new demand) with the objective of minimizing the number of used servers and, therefore, the power consumption. Commercial tools for capacity planning, such as NetIQ PlateSpin Recon [4], VMware Capacity Planner [6] and IBM Workload Deployer [3] also focus on maximizing the resource usage and power consumption savings. None of these tools explain how VM placement is carried out.

Open-source tools do not consider the impact of network topology and the communication patterns of applications, but they have been analyzed in many research works [10–12, 14, 15, 17, 23, 24]. For example, authors in [17] propose grouping VMs and servers into clusters, addressing VM placement for each <VM-cluster, server-cluster> pair as a Quadratic Assignment Problem (QAP). The VM clustering tries to maximize the intra-cluster communication and reduce the inter-cluster communication, but all VM-clusters are of equal size. The server set assigned to a VM-cluster is fixed. This work does not consider the energy consumed by physical servers. Authors in [15] follow a greedy heuristic approach, but they do not consider the large variety of applications that can run in the cloud. In [12] a greedy heuristic to improve the network utilization is presented, but it does not try to allocate the VMs in the minimum number of physical servers.

The energy consumption of a data center is derived from many elements including physical servers and the network infrastructure. However, most models for data center energy do not take into account the combined effects of these two elements. For example, authors in [17] do not consider the energy consumed by servers and [20] do not include the network infrastructure in the energy model.

Some authors address each the of the objectives separately (e.g. energy consumption and network usage) [22]. However, the VM placement problem can benefit from a multi-objective approach, where several objectives can be optimized at the same time.

Authors of [20] compare a single vs. multi-objective optimization approach and state that the latter is able to find the best trade-off between the total energy consumption and the network overhead.

3 Modeling the Cloud Computing Environment

Three main roles can be identified in a cloud computer environment: the provider, the client and the end-user. The *provider* is the owner of the infrastructure, the set of data centers that host resources and leases them to its *clients*. These in turn are the ones paying for the

utilization of the collection of VMs (and other resources) on which their applications run. These applications serve the requests submitted by many *end-users*. For the purpose of this paper, this is the way the three parties interact:

1. A client wants to deploy an application in the cloud. For this purpose, he will lease some resources (VMs) from an IaaS provider. The client will select a set of VM types with different capacities, including (among others) the network bandwidth. This interaction is carried out using a management API.
2. Given the set of VM types, and (if provided) the communication pattern of the particular application, the provider will decide how to allocate the VMs onto the different physical servers in the data center.
3. Once the application has been allocated and deployed on the assigned resources of the data center, it is ready to serve requests from end-users. For example, in case of a web application, end-users will send HTTP requests generated by their browsers.
4. Each end-user request arriving to an application will trigger the execution of pieces of code in one or more VMs, as well as some inter-VM communication (using part of the available bandwidth) for example to carry out a database query.

From the point of view of the provider, the interaction with the clients consists of requests to acquire/release collections of VMs, which impact on the allocation of resources done by the provider. In contrast, from the point of view of the application, the interaction with the end-users is by means of HTTP requests/responses, which act on the use of the resources on which the application is running.

This section presents several models used throughout this work to characterize and solve the VM placement problem. We need (and define) models to:

1. Describe the structure of cloud-hosted applications as a collection of VMs, and a communication pattern among them (that is, an application model)
2. Describe the arrival of end-user requests to a particular application, and the way each request triggers the utilization of the resources assigned to the application
3. Describe the structure of the data center managed by the provider, which is capable of hosting multiple applications of different clients
4. Describe the way the use of resources translates onto the use of power (that is, a power model), considering as resources CPUs (physical servers) and network equipment (switches)

3.1 Modeling Applications

A client willing to run an application in the cloud will interact with the provider, requesting a set of VMs. We assume that the application is built using a layered organization, with communication between layers. Also, we assume that the client knows the structure of his application, and that this structure is part of the resource request sent to the provider. The provider may take into account the application structure in order to make an optimal allocation.

In our model, the definition of an application requires several parameters: the number of layers (L), the number of VMs in each layer (N_i being i the layer identifier) and a matrix of the communication needs (or bandwidth, measured in Mb/s) between each pair of VMs i and j ($BW = [bw_{i,j}]$), and with the external world. For this work, we particularize this model to define two classes of web applications (see Fig. 1). Web applications are usually implemented using a three-layer architecture:

1. A load balancer that receives end-user requests and distributes them evenly along the VMs of the business layer; it may be implemented on a hardware device, or as a DNS-based redirection—thus, we do not include it in the application model.
2. A business layer that contains the logic of the application. It comprises a pool of VMs that processes the input requests (and generates the corresponding replies) redirected by the load balancer. The number of VMs at the business layer depends on the intensity of the workload generated by end-user requests.
3. A persistence layer that processes the database (DB) requirements of the application. The number of VMs in the persistence layer depends on the intensity of the workload generated by the application. A light workload can be managed by a single DB server that supports both read r and write w operations; we represent this class of

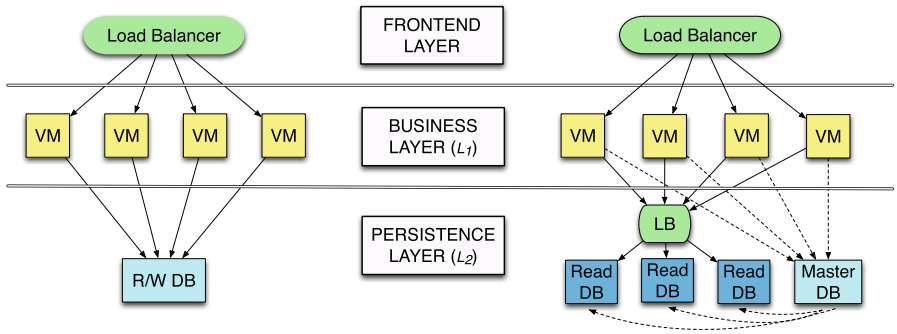


Fig. 1 Two web application types: L-WA, with a single R/W database server *(left)*; H-WA, with a master-slave database system *(right)*

applications as *L-WA*. For applications with heavy database demands (*H-WA*), a master-slave replication scheme may be applied: one of the VMs of the persistence layer is the *master* node that processes all the write w operations, while the read queries r are evenly distributed along the remaining VMs in the layer. Whenever a change (write w) is made on the master node, it is propagated to the remaining DB replicas.

IaaS providers typically offer different predefined types of VMs, with different resource sets, called *instances*. In this work, we will consider *small*, *medium* and *large* instances, with different characteristics only in terms of allotted network bandwidth (in Mb/s): bw_s= 50, bw_m=150 and bw_l=300, respectively. For *L-WA* web applications, we consider that the business layer (L_1) uses small instances and the database layer (L_2) contains a single, large VM. For *H-WA* applications, the database is modeled as a single large instance for the master DB node, and several medium-size VMs for read DB nodes.

3.2 Modeling Application Workloads

Once the application has been placed in the data center, it is ready to serve requests coming from end-users. Our model considers that a request arriving to a web application can be of one of these three types: (1) p: it is processed in the business layer, and requires no access to the database; (2) r: it requires a query (read operation) to the database; (3) w: it requires a write operation on the database.

Each request type implies different inter-VM messages and processing times in the different layers of the application. For example, an r request requires the following execution steps:

1. The initial HTTP request sent by the end-user is redirected (by the load balancer) to one of the VMs of the business layer (d_0)
2. After an initial processing time in the business layer, a query is sent to the database layer (d_1)
3. The database server will process the request (again, using some CPU time) and will send the response back to the business layer (d_2)
4. Finally, the VM in the business layer sends the response to the user, with the HTML content (d_3)

Each of the messages involved has an associated size (d_0, d_1, d_2, d_3) that is generated randomly using the ranges defined in Table 1. These values have been extracted from a report [1] that analyzes billions of web pages. The transfer rate will depend on the bandwidth allocated to each particular VM.

The response time of web applications is usually in the order of milliseconds. For our simulation-based experiments, the CPU processing time of each request is generated randomly, in the range of 50-150 ms per tier. p-type requests only require 50–150 ms in the business tier, while both r and w requests imply: 50–150 ms in the business tier, another 50–150 ms to execute the database read/write query, and 50–150 additional ms in the business tier to finally send the response back to the end-user. These values have

Table 1 Data size ranges (in KB) of each message, for each request type

Req type	d_0	d_1	d_2	d_3
p	5 – 10	0	0	5 – 500
r	5 – 10	10 – 50	20 – 400	5 – 500
w	5 – 10	10 – 50	20 – 300	5 – 500

been extracted from [8, 19]. Note that given the time-sharing nature of the CPU resource, several requests may be executed concurrently on the same VM and, thus, the processing time will increase with the degree of concurrency.

For this work we propose a workload generation function designed to generate sequences of HTTP requests, that will be used in our simulation-based experiments to "feed" applications deployed in the data center. The model is described in Algorithm 1. The generated application workload follows a diurnal pattern, with a lower average request rate at week-ends (see Fig. 2 for some examples). An initial per-day *basereq* parameter (baseline number of requests) is customized for each application, which is lowered for weekend days. Then, a sinusoidal function [13, 21] is used to generate a base number of requests per second. Using different probability distributions and ranges, we add burstiness to the workload. Finally, a request type (p, r or w) is selected randomly, together with the size of the request. In Fig. 2, we have plotted two sample application workloads that follow a daily pattern.

Algorithm 1 Per-application process to generate end-user HTTP requests

```
Data: Timestamp, basereq
Result: List of reqs (reqType, data)
reqs = 0;
r = 0;
if dayOfWeek(timestamp) is Monday-Friday then
    basereq = basereq + rnd(-basereq*0.30,
    basereq*0.30);
else
    basereq = rnd(basereq*0.30, basereq*0.50);
end
reqs = (basereq/2) * sin((2πtimestamp)/PERIOD +
4.75) + (basereq/2);
r = rand(0,1);
reqs += createBurstiness(r);
reqList = list();
foreach req do
    reqType = getReqType(appType);
    reqData = getDataSize(appType, reqType);
    add(reqList, newReq);
end
```

3.3 Describing the Data Center Structure

As previously stated, current data centers are usually built using tree-based topologies, such as fat tree and VL-2 [17]. This kind of networks are composed of several tiers of switches (we assume homogeneous switches) and several servers connected to the bottom tier of the tree (the edge or *access* tier). Each server is divided into several *slots*, where each slot can be a fraction of a core, an entire core or several cores. Application VMs are assigned to different slots of the data center servers. Throughout this work we assume that a VM consumes a slot, and that one slot is equivalent to one core of a multi-core server.

The physical configuration of a data center is defined as the number of servers (P), the number of cores (slots) per server (C_p) and the network topology.

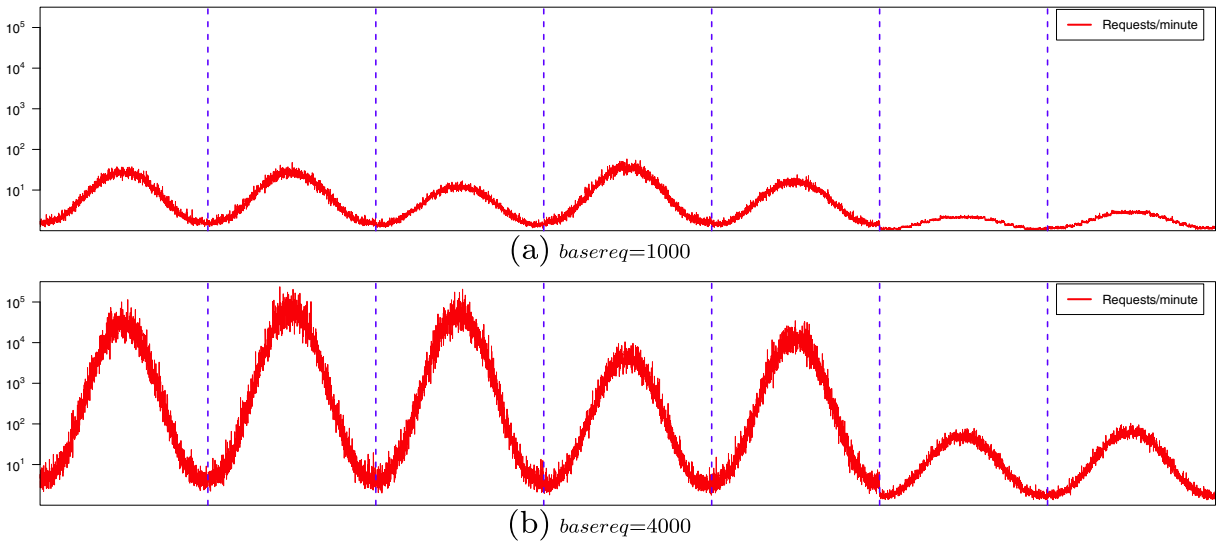


Fig. 2 Two examples of HTTP workloads generated by end-users, with different values for *basereq*

In particular, a tree-based topology is defined by the number of uplinks and downlinks of the switches (S_{up} and S_{down}), the bandwidth (Mb/s) offered by each switch port (S_{bw}) and the number of tiers of the tree (T). The communication latency between two cores i and j depends on the distance between them, measured in terms of *hops*. Given a topology, the distance $d_{i,j}$ between a any pair of cores (actually, any pair of servers) can be computed. A per-data center matrix $D = [d_{i,j}]$ summarizes all these distances.

In this work we have focused on interconnection networks built using fat trees. The main characteristics of these trees are the low average path length and the availability of multiple paths between nodes, thus performing well with almost any kind of workload. Therefore, it is the network of choice in high-performance data centers. The particular fat tree network that we have used is composed of three tiers (as depicted in Fig. 3a) with the same number of switches in each of them (see Fig. 3b). We consider that core switches are directly connected to the Internet, the bandwidth of this connection being the aggregated bandwidth of all the switches. Also, we assume that the network interfaces of the servers in the access tier are of capacity S_{bw}. In this kind of tree the distance between two cores/servers is computed as follows: cores in the same physical server are at distance 0; servers connected to the same access switch are at distance 2; if aggregation or core switches are required, the distance grows to 4 or 6 respectively.

The configuration parameters used in the experiments to model the data center are those specified in Table 2.

Table 2 Parameter configuration for the data center infrastructure

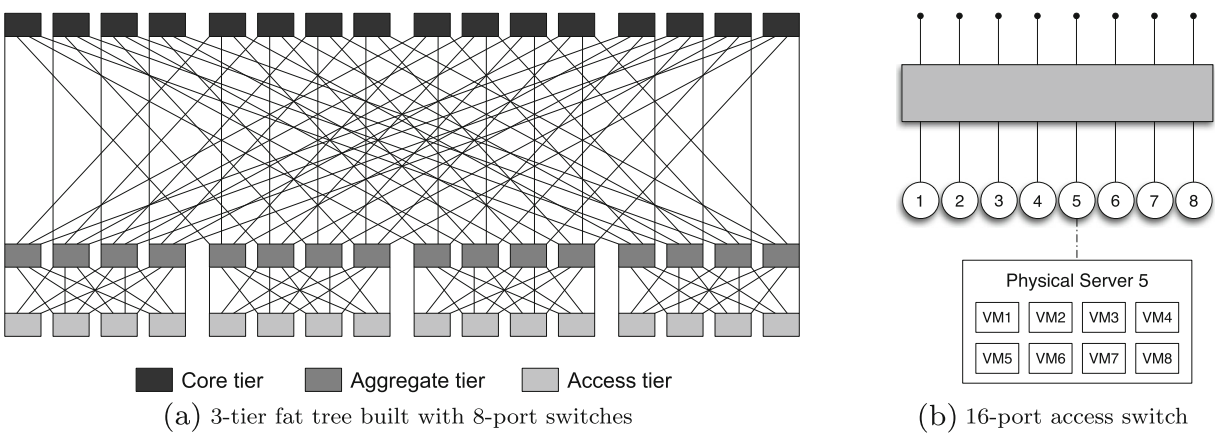
Par	Value	Description
P	512	Number of servers
C_p	8	Cores (slots) per sever
T	3	Number of tiers in the fat-tree
S_{up}	8	Number of uplink ports per switch
S_{down}	8	Number of downlink ports per switch
S_{bw}	1000	Capacity of links (Mb/s)

3.4 Modeling Power Requirements

Nowadays, the reduction of the power consumption of data centers is receiving increasing attention. Energy is consumed by servers and switches, and also by cooling and energy distribution systems. Reducing power use has direct benefits for the infrastructure provider (lowering the energy bill), while reducing the carbon footprint of the data center.

PowerNap [16] aims to reduce the consumption of unused servers by switching off memory, disk and other elements. In this work we assume that a strategy like this is used in the data center: unloaded servers and switches operate in an idle state that minimizes energy waste. We define a general model of power utilization of a device (server or switch), inspired on that provided in [16].

$$E = \begin{cases} E_{idle} & U_{active} = 0 \\ E_{active} + \frac{E_{rem} \cdot (U_{active} - 1)}{U_{total} - 1} & U_{active} > 0 \end{cases}$$



(a) 3-tier fat tree built with 8-port switches (b) 16-port access switch

Fig. 3 Representation of the physical configuration of a data center *(network and servers)*

The energy consumption E of a server/switch (in Watts) depends on the number of active cores or ports U_{active}. In the idle state, the consumption is equal to E_{idle}. The transition from the idle state to the activation of the first core/port implies an important increase in the energy utilization, because it requires turning on other resources (memory, disk) or internal fans. The activation of additional cores/ports causes a linear increase of used power. This energy model is similar to the one proposed by [18]. In this work we do not consider the penalty time due to the transition from the idle to the active state. As the authors state in [16], this time is negligible for jobs that last more than 10 ms using this model, which is actually our case.

Table 3 shows the energy consumption values used in the experiments, both for servers and switches. Values for servers are based on those in [16]. Values for switches are taken from manufacturer data sheets, and are similarly to those reported in [15].

4 Multi-objective Optimization Algorithms

This section describes the optimization algorithms used in this work to solve the problem of the initial VM placement, formulated as a multi-objective problem. We have selected three multi-objective evolutionary algorithms: NSGA-II, SPEA2 and Hype.

At each step of the optimization process (called generation), each of the algorithms maintains a set (population) of individuals (candidate solutions for a given VM placement problem). The quality of a solution is assessed using several fitness functions (objectives) that represent the (possibly constrained) problem; in this case, the placement of a collection of VMs. At each generation, the most promising individuals are chosen using a selection criterion and included in the new population. The offspring is generated by applying crossover and mutation operators over the individuals of the current population. The optimization process is repeated until the stopping criterion is fulfilled.

The result of this optimization process is a set of solutions that simultaneously optimize each of the objectives (called Pareto set). The value of the functions achieved by the Pareto optimal solutions is called Pareto front. Formally, we define a multi-objective optimization (minimization) problem subject to some restrictions as:

$$\min \left\{ f_1(x), \ldots, f_{N_{Obj}}(x) \right\} \tag{1}$$

$$\begin{cases} g_j(x) = 0 \;\; j = 1, \ldots, M, \\ h_j(x) \leq 0 \;\; j = 1, \ldots, K \end{cases} \tag{2}$$

where f_i is the $i-th$ objective function, x is a vector that represents a solution, N_{obj} is the number of objectives, $M + K$ is the number of constraints, and g_j and h_j are the constraints of the problem.

Now we explain each of the three optimization algorithms tested in this work. The main difference between them relies on the selection criterion used to choose the best candidate solutions at each generation.

- Non-dominated Sorting Genetic Algorithm II: The aim of the NSGA-II [9] algorithm is to improve the adaptive fit of a population of candidate solutions to a Pareto front constrained by a set of objective functions. The population is sorted into a hierarchy of sub-populations based on the ordering of Pareto dominance. Similarity between members of each sub-group is evaluated on the Pareto front, and the resulting groups and similarity measures are used to promote a diverse front of non-dominated solutions.
- Strength Pareto Evolutionary Algorithm 2: SPEA2 [25] uses as selection criterion a combination of the degree to which a candidate solution is dominated (strength), and an estimation of density of the Pareto front as an assigned fitness. An archive of the non-dominated set is maintained separate from the population of candidate solutions used in the evolutionary process, providing a form of elitism.

Table 3 Parameter values of energy utilization in physical servers and switches

	Consumption at	Server value (W)	Switch value (W)
E_{max}	100 % utilization	200	100
E_{idle}	Idle state	10	10
E_{active}	One active core/port	160	31
E_{rem}	Remaining $U_{active} - 1$ cores/ports	40	69

– Hypervolume Estimation Algorithm: Hype selects the best candidate solutions using the hypervolume indicator. This measure is consistent with the concept of Pareto-dominance; the set of solutions with the highest value of the indicator dominates other sets. However, the calculation of the hypervolume requires a high computational effort. Hype [7] addresses this issue trading off the accuracy of the measured estimates with the available computing resources.

5 Topology-aware Optimization

The aim of this work is to find a suitable placement for the VMs forming an application onto a set of available cores (slots) in the data center servers. Taking as starting point a VM placement obtained using a simple policy, either FF or RR, we use a bi-objective optimization algorithm to obtain a better one. This optimization takes into account the communication needs of the application being allocated, as well as the structure of the data center, aiming to benefit the cloud client (by making its application perform better) while allowing the cloud provider to save energy costs. In this section we focus on the formal definition of this particular optimization problem, as well as on the specific crossover and mutation operators needed by the three multi-objective optimization algorithms being evaluated.

5.1 Problem Definition

Given an application A with a VM set V of size N, and a subset of available cores $C' \subset C$, where C is the whole set of cores in the data center (note that usually $|C'| >> N$), the VM placement problem involves finding a mapping function φ that assigns each VM, $v \in V$ to a core $c \in C'$:

$$\varphi : V \rightarrow C'$$
$$v \mapsto \varphi(v) = c$$

A solution to the VM placement problem has the form $s = (s(1), \ldots, s(N)) = (c_1, c_2, \ldots, c_N)$ representing that the VM i is assigned to core $s(i) = c_i$.

Two major selection criteria will be considered to choose a VM placement. First, we favor solutions that minimize communication latency. For this reason, the VM placement will try to allocate the most communicative VMs onto close cores, in terms of network distance. The second criterion focuses on reducing the number of servers allocated to the application. An allocation solution that fulfills the first criterion may not satisfy the second one. For example, given an application $A = \{v_1, v_2, v_3, v_4\}$ in which communication occurs between v_1-v_2 and v_3-v_4, the first criterion may place each pair of VMs on a different physical server. However, according to the second criterion, it would be better to place all the VMs in the same server. Both criteria try to positively impact the use of data center resources, by means of reducing the number of active servers and switches, but the first one specifically tries to benefit the application, optimizing its performance. Placement solutions must obey a restriction: external communication demands of all the VMs assigned to a server cannot exceed the bandwidth of its network link S_{bw}. This constraint does not take into account communication between VMs in the same server.

More formally, we describe VM placement as a bi-objective optimization problem. The first objective function to minimize is defined as follows:

$$f_1(s) : \sum_{i,j \in V}^{N} bw_{i,j} \cdot d_{s(i),s(j)} \tag{3}$$

where $d_{s(i),s(j)}$ is the distance between the cores assigned to VMs i and j, and $bw_{i,j}$ is the bandwidth required by VMs i and j.

Given the function $\sigma(c) = p$ that returns the server p to which core c belongs to, and a solution s, we define the set of active servers for this solution as $P^s = \{p | \exists i \in \{1, \ldots, N\} \text{ s.t. } \sigma(s(i)) = p\}$. The second objective function to minimize is defined as:

$$f_2(s) : |P^s| \tag{4}$$

The requirement to guarantee that the total bandwidth usage of the VMs allocated to a server does not exceed the capacity of that server's network link can be expressed as this constraint:

$$\forall p \in P^s : S_{bw} - S^p_{bw} \geq 0 \tag{5}$$

where S_{bw} is the bandwidth of a server's link, and S^p_{bw} is the reserved bandwidth of server i, considering the previously allocated applications and also the new one.

5.2 Problem-specific Operators

As stated before, at each generation the optimization algorithms must make the current population evolve using crossover and mutation operators. In this work, we have developed specific operators that consider the characteristics of the VM placement problem.

5.2.1 Guided Crossover Operator

Crossover is applied with probability p_{cross}. It combines two individuals to generate a new one, taking into consideration the specific characteristics of the problem. Given two parents s_1 and s_2, the crossover operator generates a new child ch as follows. We define $\phi(i, s)$ as the communication cost of VM i in a candidate solution s, considering all the destinations with which it communicates, the corresponding input/output bandwidths, and the distances:

$$\forall i \in \{1, \ldots, N\} : \phi(i, s) = \sum_{j=1, j \neq i}^{N} (bw_{i,j} + bw_{j,i}) \cdot d_{s(i),s(j)} \tag{6}$$

Child ch will be constructed taking from the parents those cores that cause the lowest communication cost. That is, for each VM i, if $\phi(i, s_1) < \phi(i, s_2)$, then core $s_1(i)$ is assigned to VM i of child ch. A correction step to remove any possibly repeated position (cores) of each child may be required. In that case, the repeated core will be replaced with one of the non-used cores from one of the parents.

5.2.2 Guided Mutation Operator

Mutation is applied with a probability p_{mut}. There are two types of mutation, and the one to apply is selected based on another probability p_{mtype}. The first type performs a simple swap between any two elements of the chosen solution, without considering cores in the same server, because this change would not affect the values of the objective functions. With probability 1-p_{mtype}, the second type of mutation is applied: one of the cores assigned to the solution is replaced with a core $c \in C'$, selected randomly from the free ones using a distance-based distribution that favors physically close cores.

5.2.3 Selection Criterion for a Solution in the Pareto Front

The bi-objective optimization algorithm generates a collection of solutions for a given application (Pareto set), with different trade-offs between locality and number of allocated servers. As all Pareto optimal solutions are considered equally good, a selection criterion is required to choose one. We select the solution that is most beneficial for the provider: one that minimizes the *global* number of active servers in the data center P_{active}. Note that this criterion is completely different from the one used in f_2. With f_2 we try to minimize the number of servers assigned to *a particular* application, while here we select the solution that achieves the lowest number of active servers in *the whole data center*.

6 Experimental Framework

This section presents the simulation-based framework used to evaluate the VM placement strategies. The experiments try to provide answers to two questions: (1) which optimization algorithm performs the best when applied to the VM placement problem? and (2) what is the benefit of a network-aware multi-objective VM placement, in terms of resource usage and energy consumption?

6.1 Simulation Environment

The VM placement has been evaluated using an in-house developed simulator. This tool is able to simulate the dynamics of a realistic data center, using the models described in Section 3 for the data center topology, applications, workload generation and power consumption.

For each new application allocation request (resource acquisition request from the cloud client) arriving to the provider, the best initial placement of the collection of VMs has to be computed. A preliminary placement is generated using a simple VM placement policy: FF which searches free cores

sequentially, always starting at the first one, or RR which also performs a sequential search but starting from the last core used in the previous placement. Afterwards, the preliminary placement is improved using an optimization algorithm.

The data center is initially empty: none of the resources are reserved for any application. Sequences of acquisition/release operations (new applications, applications that end) are generated in order to populate the data center. Depending on the rate of these operations, we deal with three different load scenarios: *low* (25 %), *medium* (50 %) and *high* (75 %). These percentages indicate the average use of data center resources caused by the corresponding load.

Each experiment carried out on the simulator is divided into two phases. The first one is a warming-up phase, not used for measurements, that ends when the target load of the data center is reached and the system arrives to a steady state. The second phase, with the system in steady state, is that used for gathering metrics. It consists of ten batches with sets of 1000 operations (equally distributed between arrivals and departures). The simulator collects a variety of per-batch metrics. We have performed five repetitions of each experiment, using the same list of operations, and summarized in several tables the averaged metrics of the five repetitions.

6.2 Experiments to Compare Optimization Algorithms

Our first task was to identify the best multi-objective algorithm for our placement problem, from a set of three candidates: NSGA-2, SPEA-2 and Hype. To do so, we carried out a collection of 120 experiments in a *simplified* environment, i.e. the data center implements the acquisitions and releases of sets of VMs as requested by the cloud clients, but the dynamics of the deployed applications is not simulated: they do not execute end-user HTTP requests. For this evaluation we need to use two models: a description of the data center topology and a description of the structure (and communication needs) of the application. With this set-up, we can determine which algorithm is the one providing better values for the objective functions to be optimized, and for the criterion used to choose a solution from the Pareto set. The parameter configuration for the optimization algorithms is detailed in Table 4. Note that for this work we have not made any particular effort to tune the parameters, and we have used the same values with all the optimization algorithms.

6.3 Experiments to Evaluate VM Placement in Realistic Scenarios

Once a good optimization algorithm for VM placement has been identified, we can carry out more complex experiments, which involve the interaction of all the parties of a cloud computing set-up, in a *detailed* simulation: we simulate the arrival of acquisition/release of applications from cloud clients (which trigger operations for VM allocation/release by the infrastructure manager), and we also simulate the arrival of HTTP requests from end-users to the applications deployed in the cloud infrastructure (which trigger the use of resources in the VMs running the applications and, consequently, on the servers and attached switches). In order to implement this, we need to use a data center load model (low, medium or high), a per-application model of its HTTP workload, and a per-application model of resource utilization. As in the previous set-up, we have carried out 120 simulations.

Notice that the infrastructure manager will run a placement algorithm each time an application allocation request is received, and this can be done in a straightforward way (plain FF or RR) or in combination with an optimization algorithm. In this set of experiments we will consider only the multi-objective optimization algorithm which provides the best results in the previous set of experiments.

Table 4 Parameter configuration for the optimization algorithms (NSGA-2, SPEA-2 and Hype)

Parameter	Value	Description
N_{pop}	100	Number of individuals per generation
N_{gen}	100	Number of generations
p_{cross}	0.8	Probability of crossing operator
p_{mut}	0.8	Probability of mutation operator
p_{mtype}	0.5	Probability for mutation type

All this simulation set-up will allow us to assess the effectiveness of placement policies in terms of application efficiency (time to process HTTP requests), resource utilization and energy consumption. Notice that these metrics reflect more closely the actual needs of cloud users and providers.

7 Analysis of Results

In this section we summarize and analyze the results of the experiments reported in Sections 6.2 and 6.3.

7.1 Simplified Experiments: Evaluation of Optimization Algorithms

The results of the simplified experiments allow us to compare the simple placement policies, FF and RR, with the same ones enhanced with a topology-aware optimization, performed with NSGA-II, SPEA-2 and Hype. The first question to ask is whether or not optimization is effective (applications and data center provider benefit from it). Then, if the first answer is affirmative, the best optimization strategy must be selected.

Results are summarized in Table 5, which gathers the mean μ and standard deviation σ of the multiple simulation runs for both objective functions, f_1 (communications locality) and f_2 (number of servers assigned to the application). The number of total active servers in the data center P_{active} (P_a for short) is also included in the table.

If we focus on non-optimized policies (non-opt), clearly RR is better for applications, as it provides lower communication costs than FF in all scenarios (see f_1 values), while simultaneously providing better (smaller) f_2 values (number of servers per application). However, FF uses on average substantially fewer servers than RR. The most relevant result, though, is that applying optimization *always* provides better values for both objective functions, compared to the baseline, non-optimized FF and RR.

It is not easy to decide which optimization algorithm performs the best. We should consider f_1, f_2 as well as the number of active servers P_{active} used as the Pareto selection criterion. Attending to P_{active} values, non-optimized FF and RR almost always use more servers than the optimized counterparts (the single exception is NSGA-II-optimized RR for high loads). Among the optimized placements, SPEA-2 is, in a

Table 5 Means and deviations of values of objective functions f_1 and f_2 and number of active servers P_a for non-optimized FF and RR ($non-opt$), and for FF and RR with an additional optimization step (using NSGA-II, SPEA-2 and Hype). Results computed for *High*, *Medium* and *Low* data center loads

		First fit					Round Robin				
		μ_{f_1}	σ_{f_1}	μ_{f_2}	σ_{f_2}	μ_{P_a}	μ_{f_1}	σ_{f_1}	μ_{f_2}	σ_{f_2}	μ_{P_a}
High	non-opt	731865.40	102999.24	37.23	3.73	354.10	702092.96	113376.77	32.33	4.25	454.53
	NSGA-II	621818.21	95383.95	35.71	2.91	341.87	699406.83	107713.29	32.15	3.36	456.46
	SPEA-2	649312.15	89880.09	35.18	3.33	339.47	660725.75	65739.05	30.04	2.72	438.10
	Hype	671846.80	115480.83	36.53	3.99	353.79	697650.12	107674.54	30.91	2.94	449.66
Medium	non-opt	722702.32	107985.19	33.53	3.44	228.65	645590.51	107245.73	21.65	2.97	303.00
	NSGA-II	607612.69	101509.97	31.52	2.34	227.98	592405.23	112321.56	19.94	2.36	283.41
	SPEA-2	630735.40	106386.83	31.69	2.54	227.57	569869.03	104114.19	20.00	2.08	286.05
	Hype	600308.65	112778.65	32.06	2.70	227.59	573890.04	120745.76	20.08	2.19	287.77
Low	non-opt	690180.40	135298.79	27.35	2.63	111.15	604539.18	114483.78	17.31	1.64	125.37
	NSGA-II	574469.29	159677.89	24.72	2.19	108.05	539529.24	145097.25	16.42	1.52	109.12
	SPEA-2	559766.55	167074.10	24.52	2.39	96.75	468476.31	115624.85	15.75	1.49	112.88
	Hype	580084.18	132498.60	25.51	2.50	104.63	509938.20	131754.41	15.88	1.58	114.21

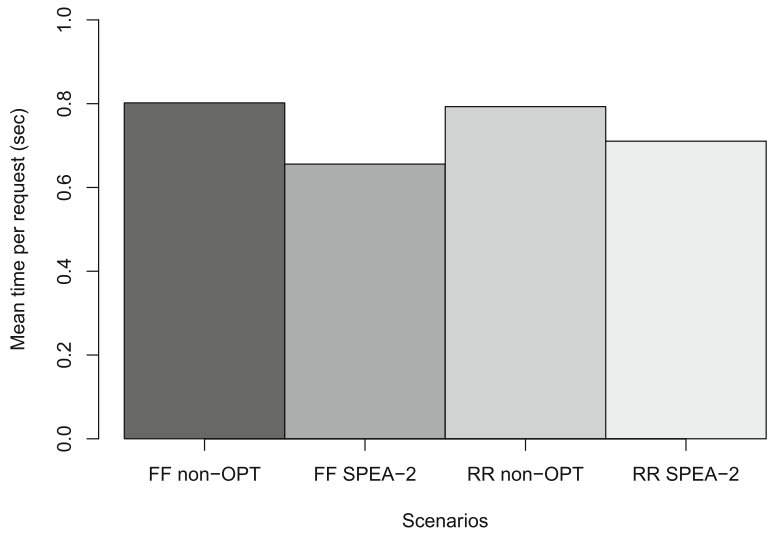


Fig. 4 Mean execution time per HTTP request *(in seconds)* for the data center with high load, comparing FF and RR without and with an additional optimization with SPEA-2

majority of cases, the one using the smallest number of servers. Furthermore, in most cases it also obtains the lowest values of objective functions f_1 and f_2.

Note that the criteria used to choose an optimization algorithm only provide *hints* about the expected benefits for applications (clients) and data center providers. The achievable benefits, expressed in more tangible terms, are analyzed in the following section.

7.2 Detailed Experiments: Evaluation of the Benefits of Optimization-based Placement

The objective functions f_1 and f_2, together with the Pareto selection criterion, were designed to have a positive impact on both the applications and the data center, but we need to assess those impacts in a meaningful, measurable way. For applications, we want to know the improvements in terms of the mean execution time per HTTP request. For the data center, we are interested in achieved energy savings. For the latter we could also provide the number of active servers and switches, but energy provides a single metric that assesses how "green" our proposals are.

Figure 4 compares the HTTP request processing time for FF and RR without and with the optimization step carried out with SPEA-2. The figure is only for the highly-loaded data center, but values for other loads are similar because the (simulated) data center never does over-subscription: it is guaranteed that clients use the resources they pay for in an exclusive

Table 6 Energy consumption (in MWatts/hour)used by physical servers, switches and total, for non-optimized $Non-opt$ and SPEA-2 placements, for different data center loads

		First fit			Round Robin		
		E_{server}	E_{switch}	E_{total}	E_{server}	E_{switch}	E_{total}
High	non-opt	69.97	24.41	94.38	73.54	24.83	98.37
	SPEA-2	62.85	24.68	87.53	59.87	24.92	84.79
Medium	non-opt	45.05	25.43	70.49	70.82	25.54	96.37
	SPEA-2	42.61	25.60	68.20	49.12	25.36	74.48
Low	non-opt	43.80	26.21	70.02	68.24	26.28	94.52
	SPEA-2	42.00	26.17	68.17	45.81	26.21	72.02

way. However, the placement policy has an impact on the inter-VM communication cost: a good placement reduces inter-VM distance and, thus, HTTP requests are performed faster. We have measured 656 vs. 802 ms for FF (optimized vs. non-optimized) and 710 vs. 793 ms for RR.

Regarding the energy consumed by the data center, we have to consider that FF-based policies (with or without optimization) always use fewer servers than the RR-based alternatives. Simultaneously, all the tested policies make *very similar* use of network links and switches, because of the fat-tree topology that distributes data movements along all upper-level switches: locality only translates into negligible gains at the access layer. This comes at no surprise because the locality function to optimize is used on a per-application basis, not taking into consideration the global use of the network. The combined result is that the placement policy using fewer servers is the one consuming less power, as reflected in the measurements summarized in Table 6. Plain RR is more power-hungry than FF. When SPEA-2 is put to work, differences between these two strategies blur. Optimized FF is in most cases still superior, but RR is the best choice for highly loaded data centers.

8 Conclusions and Future Work

Throughout this paper we have demonstrated that an IaaS provider can improve the VM placement policy in use by applying an optimization strategy, achieving benefits not only for the provider but also for clients and end-users. Furthermore, this optimization can be done at a negligible cost: it is applied when allocating a new application, taking only a few seconds. Benefits for the provider are measured in terms of used servers and switches, and immediately translate into reduced power demands (resulting in a "greener" data center). Benefits for the applications are achieved by improving their ability to process user requests. As a direct effect of improving communication latencies, the average execution time per request is reduced (up to 11–19 %). Thus, the cloud client (that is, the application owner) will be better able to comply with QoS expected by end users. Simultaneously, using optimization we are able to improve the number of active servers in the data center and, therefore, the overall energy consumed: for highly loaded data centers, savings range between 7.26–13.81 %. Globally, the best tested placement strategy is FF with SPEA-2 optimization, although in some instances (optimized) RR yields better results.

The utilization of this optimization-based placement requires an effort from the clients: they must specify, or at least provide some hints about, the application architecture and its communication needs. This is an old and well-known problem: application dimensioning. Our proposal also requires including, in the decision processes for data center resource management, knowledge about the structure of its interconnection network.

This work is focused on the initial VM placement of applications. However, the state of a data center is very dynamic: new applications arrive, other applications are removed, and deployed applications can increase/reduce the resources allocated to it. After all, one of the key characteristics of cloud computing is elasticity. Most probably, after some time, the initial VM placement may become sub-optimal (for example, using an excessive number of active servers), and reallocation of some VMs could be necessary (for example, to consolidate more VMs in fewer servers to obtain a smaller set of active servers). This process is called re-consolidation. We plan to address re-consolidation as a multi-objective optimization problem, trading off its benefits with its costs. Each VM migration should be planned carefully as it causes extra CPU costs and network overhead that may negatively affect the allocated applications.

Elasticity allows the applications to dynamically scale the acquired resources (the number of VMs in horizontal scaling) depending on the input workload. Thus, the number of VMs will vary with time and the infrastructure provider should be able to optimize not only the initial placement, but also the addition of new VMs. We plan to adapt our proposal to deal with scalable applications.

Providers usually over-subscribe resources: users rarely exploit 100 % of the assigned resources (including cores, memory, network bandwidth, etc.) Therefore, it is common practice to assign to a server "extra" slots. However, providers have to monitor resource usage carefully, to ensure that the aggregated current demands do not exceed server capacity: this would be very negative in terms of the QoS perceived by clients.

The QoS could be easily fitted as an extra objective in a multi-objective optimization formulation of the VM placement and/or re-consolidation.

Finally, we plan to implement optimization-based VM placements as part of an open source cloud infrastructure managing software package, in order to better assess its real-world applicability.

Acknowledgments This work has been partially supported by the Saiotek and Research Groups 2013–2018 (IT-609-13) programs (Basque Government), TIN2013–41272P and COMBIOMED network in computational biomedicine (Carlos III Health Institute). Dr. Pascual is supported by a postdoctoral grant of the UPV/EHU. Mrs Lorido-Botran is supported by a doctoral grant from the Basque Government. Prof. Miguel-Alonso is a member of the HiPEAC European Network of Excellence.

References

1. Google report (2010). https://developers.google.com/speed/articles/web-metrics
2. Eucalyptus (2014). http://www.eucalyptus.com/, [Online; accessed 6-June-2014]
3. IBM. [Online; accessed 6-June-2014] (2014). www.ibm.com/software/products/us/en/workload-deployer
4. NetIQ. [Online; accessed 6-June-2014] (2014). https://www.netiq.com/products/recon/
5. OpenNebula. [Online; accessed 6-June-2014] (2014). http://opennebula.org/
6. VMware. [Online; accessed 6-June-2014] (2014). http://www.vmware.com/products/capacity-planner/
7. Bader, J., Zitzler, E.: Hype: An algorithm for fast Hypervolume-based Many-objective optimization. Evol. Comput. **19**(1), 45–76 (2011)
8. Bi, J., Zhu, Z., Tian, R., Wang, Q.: Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. In: Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, pp. 370–377 (2010). doi:10.1109/CLOUD.2010.53
9. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
10. Fan, P., Chen, Z., Wang, J., Zheng, Z.: Online Optimization of VM Deployment in IaaS Cloud. In: ICPADS, pp. 760–765 (2012)
11. Fang, W., Liang, X., Li, S., Chiaraviglio, L., Xiong, N.: VMPlanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers. Comput. Net. **57**(1), 179–196 (2013). doi:10.1016/j.comnet.2012.09.008. http://www.sciencedirect.com/science/article/pii/S1389128612003301
12. Georgiou, S., Tsakalozos, K., Delis, A.: Exploiting Network-Topology Awareness for VM Placement in IaaS Clouds. In: CGC, pp. 151–158 (2013)
13. Islam, S., Lee, K., Fekete, A., Liu, A.: How a Consumer Can Measure Elasticity for Cloud Platforms. In: Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, ACM, New York, NY, USA, ICPE '12, pp. 85–96 (2012). doi:10.1145/2188286.2188301
14. Kliazovich, D., Bouvry, P., Khan, S.: DENS: data center energy-efficient network-aware scheduling. Cluster Comput. **16**(1), 65–75 (2013). doi:10.1007/s10586-011-0177-4
15. Mann, V., Kumar, A., Dutta, P., Kalyanaraman, S.: VMFlow: leveraging VM mobility to reduce network power costs in data centers. In: In: NETWORKING, vol. I, pp. 198–211 (2011)
16. Meisner, D., Gold, B., Wenisch, T.: PowerNap: eliminating server idle power. ACM SIGPLAN Notices **44**(3), 205–216 (2009)
17. Meng, X., Pappas, V., Zhang, L.: Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement. In: IEEE INFOCOM, pp. 1154–1162 (2010)
18. Reviriego, P., Sivaraman, V., Zhao, Z., Maestro J.A., Vishwanath, A., Sanchez-Macian, A., Russell, C.: An energy consumption model for Energy Efficient Ethernet switches. In: High Performance Computing and Simulation (HPCS), 2012 International Conference on, pp. 98–104 (2012)
19. Shen, Z., Subbiah, S., Gu, X., Wilkes, J.: Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In: Proceedings of the 2Nd ACM Symposium on Cloud Computing, ACM, New York, NY, USA, SOCC '11, pp. 5:1–5:14 (2011). doi:10.1145/2038916.2038921
20. Tziritas, N., Xu, C.Z., Loukopoulos, T., Khan, S.U., Yu, Z.: Application-Aware Workload Consolidation to Minimize Both Energy Consumption and Network Load in Cloud Environments. In: Parallel Processing (ICPP), 2013 42nd International Conference on, pp. 449–457 (2013)
21. Urdaneta, G., Pierre, G., van Steen, M.: Wikipedia workload analysis for decentralized hosting. Comput. Net. **53**(11), 1830–1845 (2009)
22. Wang, S.H., Huang, P.W., Wen, C.P., Wang, L.C.: EQVMP: Energy-efficient and QoS-aware virtual machine placement for software defined datacenter networks. In: Information Networking (ICOIN), 2014 International Conference on, pp. 220–225 (2014)
23. Wo, T., Sun, Q., Li, B., Hu, C.: Overbooking-Based Resource Allocation in Virtualized Data Center. In: ISORCW, pp. 142–149 (2012)
24. Yapicioglu, T., Oktug, S.: A Traffic-Aware Virtual Machine Placement Method for Cloud Data Centers. In: Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, IEEE Computer Society, Washington, DC, USA, UCC '13, pp. 299–301 (2013)
25. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In: Giannakoglou, K.C., Tsahalis, D.T., Periaux, J., Papaliliou, K.D., Fogarty, T. (eds.), pp. 95–100. Barcelona, Spain (2002)