Decentralized and Energy-Efficient Workload Management in Enterprise Clouds

Michael Pantazoglou, Gavriil Tzortzakis, and Alex Delis

Abstract—We present a decentralized approach towards scalable and energy-efficient management of virtual machine (VM) instances that are provisioned by large, enterprise clouds. In our approach, the computation resources of the data center are effectively organized into a hypercube structure. The hypercube seamlessly scales up and down as resources are either added or removed in response to changes in the number of provisioned VM instances. Without supervision from any central components, each compute node operates autonomously and manages its own workload by applying a set of distributed load balancing rules and algorithms. On one hand, underutilized nodes attempt to shift their workload to their hypercube neighbors and switch off. On the other, overutilized nodes attempt to migrate a subset of their VM instances so as to reduce their power consumption and prevent degradation of their own resources, which in turn may lead to SLA violations. In both cases, the compute nodes in our approach do not overload their counterparts in order to improve their own energy footprint. An evaluation and comparative study of the proposed approach provides evidence of its merits in terms of elasticity, energy efficiency, and scalability, as well as of its feasibility in the presence of high workload rates.

Index Terms—K.6.4.a Centralization/decentralization, H.3.4.b Distributed systems, B.9.2 Energy-aware systems

1 Introduction

THE continuing growth of cloud computing [1] [2] and its immediate uptake by the industry have yielded a large number of cloud-based services spanning the infrastructure, platform, and software levels. However, as the demand for cloud services continues to increase at a global scale, so does the energy consumption of the service providers' data centers and, ultimately, their negative impact on the environment. Nowadays, enterprise-scale cloud computing infrastructures consume huge amounts of electrical energy, contributing to high operational costs and carbon footprints to the environment [3]. Still, the majority of enterprise cloud data centers utilize only a fraction of their available resources, while a considerable part of their power consumption is lost due to both over-provisioned and idle resources [4]. It therefore becomes important for cloud service providers to adopt appropriate measures in order to attain energy-efficient processing and utilization of their computing infrastructure.

In computation–intended data centers, the workload gets essentially translated into a number of provisioned virtual machine (VM) instances. To address the aforementioned problems in such settings, the technology of dynamic VM consolidation has been devised [5], widely studied, and applied [6] [7] [8]. In a nutshell, dynamic VM consolidation continuously strives to reduce the energy consumption of the data center by packing the running VM instances to as few physical machines as possible, and consequently switching off the unnecessary resources. Combined with the use of live VM migration [9], [10], which refers to the process of moving a running VM instance between different physical compute nodes without disconnecting the client, VM consolidation has become feasible in terms of cost [11] [12], and it can considerably improve the energy footprint of cloud data centers.

Still, in the presence of enterprise clouds consisting of hundreds to even thousands of physical machines utilized for the provision of large numbers of VM instances, energy-efficient load balancing through VM consolidation becomes a challenging task. Indeed, the problem of VM consolidation is an applied form of bin packing, which is by nature a combinatorial *NP*-hard problem [13] and therefore, expensive to compute when large numbers of physical machines and thousands of VM instances are involved. To date, most of the existing approaches rely on centralized [14], hierarchical [15], or ring [16] topologies, all of which exhibit certain performance limitations as both data centers and their workload scale out. Consequently, it is critical for cloud service providers to select an appropriate and scalable data center architecture in order to carry out the VM consolidation process in an efficient way.

1.1 Contribution

To support energy-aware workload management in enterprise clouds and overcome the inherent scalability and performance limitations of centralized and hierarchical approaches, we propose a fully decentralized and energy-efficient load balancing approach. Our approach implements dynamic VM consolidation and relies on live VM migration. Specifically, the physical machines of the data center that are used to host the VM instances are effectively self-organized in a highly scalable peer-to-peer overlay network. Each physical machine is allowed to operate autonomously and manage its own workload which is expressed as the number of locally hosted VM instances. In this context, the overall objective of our distributed load balancer is to maintain as few physical machines as needed for provisioning the current workload of the data center. Empowered by

• *The authors are with the Department of Informatics and Telecommunications, University of Athens, Panepistimiopolis, 15784 Athens, Greece. E-mail: {michaelp, gtzortzakis, ad}@di.uoa.gr*

our suggested distributed algorithms, all physical machines cooperatively contribute towards achieving this shared goal, without supervision and/or help from any central components. Most importantly, physical machines are not aggressively switched off as this could lead to overloading of their counterparts, and subsequently cause SLA violations.

The salient features of our approach are:

- *Decentralization:* our load balancer is fully decentralized, and operates through distributed algorithms that are carried out by each physical machine individually. The peer-to-peer overlay is considerably flexible and efficient, and as such, our approach can easily scale to large data centers.
- *Elasticity:* the data center's size, i.e., the number of active physical machines, is scaled up and down as the overall workload is increased and decreased. Physical machines are authonomously switched on and off depending on the number of VM instances leading to enhanced efficiency in terms of the overall energy consumption.
- *Cost effectiveness:* The protocols for the maintenance of the peer-to-peer topology exhibit an $O(log_2N)$ complexity, with N being the number of physical machines. Hence, their additional operational cost is insignificant compared to the overall energy costs of running the data center. Moreover, our load balancer dynamically consolidates the VM instances and reduces the number of turned on physical machines, while at the same time, it tries to keep the number of overutilized physical machines as low as possible.

1.2 Paper Structure

In the following section, we present an analysis of the relative literature and pinpoint the added value of our work in the context of energy-efficient workload management in private, enterprise clouds. Then, we proceed in Section 3 with the detailed presentation of our proposed approach. An experimental, simulation-based evaluation and comparative study of our approach along with the retrieved measurements are presented and discussed in Section 4, before we conclude this paper and identify paths for future work in Section 5.

2 Related Work

Green Cloud computing [17] [18] through VM consolidation is a relatively new research topic that has gained considerable momentum. It has however received extensive attention in the last few years as data center operators struggle to minimize their energy consumption and thereby, their operational costs. In this section, we survey related work in the field of energy-efficient load balancing for private cloud environments so as to appropriately position our approach and its contribution.

In their recent work [19], Sampaio and Barbosa propose a mechanism for the dynamic consolidation of VMs in as few physical machines (PM) as possible; the aim is to reduce the consumed energy of a private cloud without jeopardizing the compute nodes reliability. The approach is implemented via a sliding-window condition detection mechanism and relies on the use of a centralized cloud manager that carries out the VM-to-PM mappings based on periodically collected information.

The *ecoCloud* approach, proposed by Mastroianni et al. [20], is another effort for power-efficient VM consolidation. In ecoCloud, the placement and migration of VM instances are driven by probabilistic processes considering both, the CPU and RAM utilization. *ecoCloud* enables load balancing decisions to be taken based on local information, although the framework still relies on a central data center manager for the coordination of the VM host servers.

Beloglazov and Buyya described the architecture and specification of an energy efficient resource management system for virtualized cloud data centers in their past work [21]. The presented system is semi-decentralized as it has a hierarchical architecture, while VM consolidation is performed through a distributed solution of the bin-packing problem. In another, more recent work [22], Beloglazov et al. present a set of algorithms for energy-efficient mapping of VMs to suitable cloud resources in addition to dynamic consolidation of VM resource partitions. The algorithms are implemented by a *Green Cloud computing infrastructure*, which introduces an additional layer to the typical cloud architecture. This infrastructure comprises a set of centralized components, among which are the i) *energy monitor* which observes energy consumption caused by VMs and physical machines, and ii) *VM manager* which is in charge of provisioning new VMs as well as reallocating VMs across physical machines on the basis of the information collected by the energy monitor.

SCORCH, proposed by Dougherty et al. [23], is a model-driven approach for optimizing the configuration, energy consumption, and operating cost of cloud infrastructures. In this approach, energy efficiency is sought through the use of a shared queue containing prebooted and preconfigured VM instances that can be rapidly provisioned. *SCORCH* introduces models to represent the configuration requirements of multiple software applications and the power consumption/operational costs when using different VM configurations. Those models are transformed into constraint-satisfaction problems, which are analyzed to determine the set of VMs that simultaneously maximizes the auto-scaling queue hit rate and minimizes its energy consumption.

Bruneo et al. presented a performance analysis framework, based on stochastic reward nets, that is able to implement resource allocation strategies in a green cloud infrastructure [24]. A series of evaluation results give evidence of the effectiveness of the proposed strategies. Besides, Lee and Zomaya presented two energy-conscious task consolidation heuristics, which aim to maximize resource utilization and explicitly take into account both active and idle energy consumption [25]. Those heuristics can help mapping each VM to the physical machine on which the energy consumption for its execution is explicitly or implicitly minimized without the performance degradation of that task. In both aforementioned works, it is not clarified whether the proposed solutions can be implemented in a decentralized manner or not.

In an effort to constrain the frequent switching on/off of the compute nodes in a cloud, and thereby reduce the corresponding power consumption, Li et al. proposed a demand forecast approach [26] that could be used in conjunction with dynamic VM consolidation solutions. The proposed approach allocates a VM instance to the appropriate physical machine based on the variant number of client requests over time.

Mazzucco et al. have proposed and evaluated energy-aware allocation policies that aim to maximize the average revenue received by the cloud provider per time unit [27]. This is achieved by improving the utilization of the server farm through powering off excess servers. The proposed policies are based on dynamic estimates of user demand, and models of system behavior. It is also shown that these policies perform well under different traffic conditions.

Another approach based on live VM migration for energy savings in the Eucalyptus cloud computing environment [28] was proposed by Graubner et al [29]. In this approach, a hierarchical organization of the data center components is required, whereby a *cluster* controller coordinates with the *node* controllers in order to resolve the VM relocations.

V-MAN, proposed by Marzolla et al. [30], is among the very few decentralized approaches to VM consolidation in large cloud data centers. *V-MAN* employs the *NEWSCAST* gossip protocol [31] on top of an unstructured peer-to-peer overlay. In addition, it can quickly converge with an almost-optimal VM placement starting from arbitrary initial VM allocations. Thus, *V-MAN* reduces the number of active hosts and improves the energy footprint of the data center. In the *V-MAN* topology, nodes obtain a local view and aware only of a subset of the data center's resources. *V-MAN* does not consider the VM migration costs, while its load balancer is driven exclusively by the number of VM instances hosted by each node. The latter is not always proportional to the power consumed. Furthermore, in its continuous effort to consolidate the VM instances within the smallest possible number of active nodes, *V-MAN* appears to not consider the potential overuse of active nodes resources which may lead to SLA violations. In this context, *V-MAN* would be unable to effectively handle situations where the workload is progressively increased for long periods of time. In contrast, our suggested approach ensures that no compute node will be switched off at the expense of overloading other nodes.

In bio-inspired approach [32], a self-organizing algorithm to redistribute load among servers is proposed by Barbagallo et al. The algorithm operates in a way reminiscent to swarms of birds or insects: each server node creates a scout, which then travels from one node to the other, following the established links between them. Each scout explores the visited servers and compares their status to the original one. Subsequently, based on the collected information and a probability distribution, the server can decide whether to migrate a subset of its currently provisioning VM instances to the visited servers, or not. Similarly to our work, the goal of this approach is to bring the network from a state of load randomly distributed among nodes to a situation where portion of the servers are used with their maximum efficiency and the remaining are switched off. Unlike our approach, the supported model defines a binary state for nodes –*on* or *hibernated*– and thus, it does not consider the *overutilized* state. Moreover, the approach works under the assumption that the servers are somehow grouped and connected together. By and large, this could entail the risk of server nodes ending up as *islands* without the ability to connect and communicate –essentially migrate VMs– with other nodes in the data center. In contrast, our hypercube-based approach ensures that, at any given time and in all circumstances, all compute nodes cooperatively partake in the creation of a "global" load-balancing strategy.

In a recent work by Hellerstein [33], the *Harmony* resource monitoring and management system is employed in a data center to classify and cluster heterogeneous tasks (VMs), which are then distributed to the available, potentially heterogeneous compute nodes, in an energy-aware fashion. *Harmony* does not impose any specific topology on the compute nodes, as it is implied that the optimization algorithms are executed by a centralized task scheduler. Nevertheless, the proposed scheme could be combined with decentralized load-balancers such as our proposed approach or *V-MAN*, in order to enable a fine-grained selection mechanism for VM live migration.

Driven by their finding that network elements considerably contribute to the energy costs of data centers, Fang et al. proposed the *VMPlanner* approach [34] for network–power reduction in virtualization-based data centers. *VMPlanner* aims at optimizing both the virtual machine placement and the traffic flow routing so as to turn off as many unneeded network elements as possible for power saving. The whole approach is formalized as an optimization problem, which is solved by *VMPlanner* with the use of approximation algorithms.

The *VMFlow* [35] framework is another approach similar to *VMPlanner* that takes into account both the network topology and traffic in order to accomplish network power reduction while satisfying as many network consumers as possible. *VMFlow* presents the network power aware VM placement and demand routing as an optimization problem ultimately solved with the use of a heuristic. *VMPlanner* and *VMFlow* can be regarded as complementary to aforementioned techniques as well as our proposal; *VMPlanner* and *VMFlow* do not consider power consumption of compute nodes stemming out of the CPU usage.

When compared to the above reviewed approaches, our proposal deploys a fully decentralized architecture and utilizes distributed load-balancing algorithms that in conjunction empower compute nodes to operate autonomously. Thanks to those features, our approach can scale and is therefore applicable to enterprise-grade data centers that consist of possibly thousands of compute nodes provisioning large numbers of VM instances. Moreover, the hypercube protocols ensure that, the compute nodes can join and depart from the network without affecting the overlay's completeness and stability of its structure. Finally, the load balancing and VM migration decisions are taken solely based on the power consumption of the compute nodes which makes our approach independent of the VM nature and specificities.

3 The Proposed Approach

This section presents the architecture, representation model, and algorithms of our proposed approach towards decentralized and energy-efficient workload management in private cloud data centers.

3.1 Data Center Structure

A private, enterprise cloud data center typically consists of one or more physical *controller* nodes, whose purpose is to maintain the overall cloud–OS [36], and a farm of physical *compute* nodes, which are used to provision VM instances. Further, the data center usually relies on a shared storage space for storing the VM disk images. Since our goal is to enable decentralized workload management, we organize the data center's compute nodes in an n-dimensional binary hypercube topology [37]. Hypercubes particularly possess a series of attributes, which are also essential to our approach:

- *Network symmetry*: All nodes in a hypercube topology are equivalent and so, no node incorporates a more prominent position than the others.
- *Cost effectiveness*: The hypercube topology exhibits an $O(\log_2 N)$ complexity with respect to the messages that have to be sent, for a node to join or leave the network. Hence, the execution of the respective join and departure protocols does not inflict notable overheads on the overall performance of the compute nodes and the data center at large.
- *Churn resilience*: It is always possible for the hypercube topology to recover from sudden node losses, even at large scale.

By organizing the data center's compute nodes in a hypercube structure, each one of them is directly connected to at most n neighbors, while the maximum number of compute nodes is $N = 2^n$.

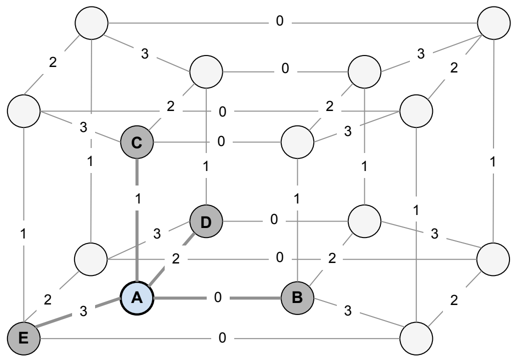

Fig. 1. Topology of a 4-dimensional binary hypercube.

For instance, in a 4-dimensional binary hypercube, the compute nodes are linked as shown in Fig. 1. Each link is given a numeric label that denotes the *dimension* in which the two linked nodes are neighbors of each other. In this example, compute node A is only connected to compute nodes B, C, D, and E, which are its neighbors in the hypercube dimensions 0, 1, 2, and 3, respectively.

The data center scales up and down as compute nodes are added and removed, according to the hypercube node-join and node-departure algorithms, which ensure that the hypercube structure will remain as complete and compact as possible [38]. In addition, we assume that the data center supports *live* VM migration, as this technique is currently supported by most major hypervisor technologies, such as Xen [39] or VMware [40], making it possible to migrate a VM instance from one compute node to another within the data center with near-zero downtime.

3.2 Compute Nodes

Compared to the other system resources of a compute node, such as memory and network, the CPU consumes the main part of its power, and its utilization is typically proportional to the overall system load [22]. Based on this fact, we focus on managing the CPU power consumption of the compute nodes in the data center. Hence, in the remainder of this paper, we will use the phrases ”CPU power consumption” and ”power consumption” interchangeably.

Each compute node in the data center is represented by a tuple as follows:

$$c = \{id, W(t), p(t), s(t), N_h, E\} \tag{1}$$

In the above tuple, id is the unique identifier of the compute node within the data center, $W(t)$ is its current *workload*, $p(t)$ is its current CPU *power consumption*, $s(t)$ is its current *state*, while $N_h = \{h_d = \{id_d, s_d(t)\}\}_{d=0}^{N-1}$ is a set that is used as the node's *cache* maintaining the identity and status of the hypercube nodes. With N_h, it is easy for the compute node to identify which of the contained nodes are its neighbors in the topology, as their respective identities differ by only one bit from its own identity. Furthermore, in the tuple c, E denotes its *power profile*, which specifies the following constant properties:

– p_{idle} defines the amount of power consumed by the compute mode when idle, i.e., when the compute node is not hosting any VM instances.
– p_{min} defines the level of power consumption, below which the compute node should try to migrate all its locally running virtual machines and shut down.
– p_{max} defines the critical level of power consumption, above which the compute node's performance is significantly degraded as its hardware resources, particularly the CPU, are over-utilized.

For the above properties, it naturally holds that:

$$p_{idle} < p_{min} < p_{max} \tag{2}$$

The current workload of a compute node amounts to the number of VMs that are locally running:

$$W = \{vm_j\}_{j=1}^{k} \tag{3}$$

In turn, each VM vm_j is represented as a tuple

$$vm_j = \{id_{vm}^j, p_{vm}^j\} \tag{4}$$

where id_{vm}^j is the VM's unique identifier within the data center, and p_{vm}^j is the VM's current CPU power consumption.

At any given time t, a VM j being hosted by compute node i introduces a CPU power overhead $p_{vm}^j(t)$. Assuming that node i is active and hosting k VMs at time t, its overall CPU power consumption is

$$p_i(t) = p_{idle}^i + \sum_{j=1}^{k} p_{vm}^j(t) \tag{5}$$

In Equation 5 we have omitted the CPU power consumption induced by the non-VM activities of the node, and the hypercube maintenance protocols, as the overall energy footprint of an active compute node is predominantly defined by the CPU utilization of its hosted VMs.

Based on its current power consumption $p_i(t)$ at any given time t, the *state* $s_i(t)$ of a compute node i takes one of the following values:

$$s_i(t) = \begin{cases} \text{switched-off} & p_i(t) = 0 \\ \text{idle} & p_i(t) = p^i_{idle} \\ \text{underutilized} & p^i_{idle} < p_i(t) \leq p^i_{min} \\ \text{ok} & p^i_{min} < p_i(t) < p^i_{max} \\ \text{overutilized} & p_i(t) \geq p^i_{max} \end{cases} \quad (6)$$

At any given time t, a compute node c is considered as *active*, if $s(t) \neq$ switched-off. All the active compute nodes in the data center periodically exchange *heartbeat* messages with their hypercube neighbors, as mandated by the hypercube protocol [37]. In our case, this inherent mechanism of the hypercube topology is further leveraged as follows. At the end of each heartbeat period, T_h, each node sends to its neighbors the delta between the last and the new contents of its maintained set, N_h. In other words, each node transmits only those entries, which correspond to nodes with a changed status, while each such entry contributes a total of $3 \cdot d$ bits in the heartbeat payload (3 bits needed for encoding the status, and d bits needed for encoding the node identity in the hypercube).

It can be easily shown that, within at most d heartbeat periods, each node in the hypercube will become aware of the statuses of all other nodes. Hence, on the basis of the suggested scheme, the compute nodes are also periodically synchronized every $T = c \cdot d \cdot T_h$ seconds, where d is the hypercube dimension, and $c > 0$ is an integer parameter set by the data center's administration to regulate the frequency of synchronization.

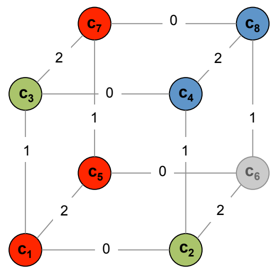

Fig. 2. A random three-dimensional hypercube.

For instance, let's consider the compute nodes of a three-dimensional hypercube of Fig. 2. In this snapshot, the red nodes (c_1, c_5, c_7) are *overutilized*, the green ones (c_2, c_3) are *ok*, the blue ones (c_4, c_8) are *underutilized*, while the grey one (c_6) is *switched-off*. For the sake of the example, we assume that the local cache, N_h, of each node is currently empty. Now, let us describe how the compute node c_1 will populate its local cache during the course of three heartbeat periods. At the end of the first period, c_1 will receive three heartbeat messages from its immediate neighbors, c_2, c_3, and c_5, which, in the worst case, will only contain their own state. Hence, at this stage, we have $N_h^{c_1} = \{c_2, c_3, c_5\}$. When the second period ends, c_1 will again receive three heartbeat messages from the same nodes as previously. This time, however, the heartbeat messages will be enriched with additional information. Specifically, the heartbeat message from c_2 will at least include the states of nodes c_4 and c_6. Similarly, the heartbeat message from c_3 will at least include the states of nodes c_4 and c_7, while the heartbeat message from c_5 will at least include the states of nodes c_7 and c_6. At this point, the local cache of c_1 has been updated with the new content: $N_h^{c_1} = \{c_2, c_3, c_4, c_5, c_6, c_7\}$.

At the end of the third heartbeat period, all heartbeat messages received by c_1 will only contain the state of node c_8, thus allowing c_1 to obtain a "*global view*" of the data center. It should be noted that, depending on the timing of exchanged heartbeat messages, it could be possible for c_1 to receive the state of c_8 already by the end of the second heartbeat period. In any case, each compute node in our example requires at most three heartbeat periods in order to fully populate its local cache.

On the basis of this representation model of the compute nodes, and given the hypercube topology in which they are organized, we present in the following subsections the details of our decentralized and energy-efficient workload management approach.

3.3 Initial VM Placement

The data center clients can request the creation and allocation of new VM instances at any time, given that the data center has not exceeded its maximum capacity, i.e., at least one of its compute nodes is not in the overutilized state. In similar fashion, VM instances can be terminated at any time. In our approach, the data center is able to initially place VM instances to its compute nodes in a completely decentralized manner, by leveraging the hypercube topology. When the provision of a new VM instance is requested by the client, an active compute node is selected by one of the data centers controller nodes, by performing a random walk within the hypercube. The selected compute node assumes the *VM initiator* role and executes the VM placement process of Algorithm 1.

The VM initiator first checks the contents of its local set, N_h, in order to verify that the data center has not reached its maximum capacity. If not, the VM initiator retrieves the nodes that are currently in the *ok* state. If no such nodes are found, the VM initiator proceeds with the retrieval of the currently underutilized nodes from N_h. Likewise, if no such nodes are found, the VM initiator retrieves any idle nodes, and, in the worst case, all nodes designated as switched-off.

Eventually, the VM initiator sorts the list of retrieved nodes in ascending order by proximity in terms of their relative position in the hypercube. Then, it extracts the first entry in order to send the new VM instance to the corresponding node. Although the target node and the VM initiator may be not directly neighbored, the latter can effectively reach the former thanks to the hypercube's shortest-path routing mechanism [41]. It should be also noted that, if the target node h is in the *ok* state, the VM instance will be sent to it only under the premise that the following condition holds true:

$$p_{vm} \geq (p^h_{max} - p^h(t)) \quad (7)$$

Algorithm 1: Initial VM placement

```
input: An active compute node (VM initiator), $c$
input: The VM instance, $vm$
result: true in success, or false in failure
begin
    $N_h \leftarrow$ get local cache of $c$
    if all nodes in $N_h$ are overutilized then
        return false
    end
    $L \leftarrow$ get all ok nodes from $N_h$
    if $L \neq \emptyset$ then
        sort $L$ in ascending order by proximity
        for $h \in L$ do
            if $p_{vm} \geq (p^h_{max} - p^h(t))$ then
                place $vm$ to $h$
                return true
            end
        end
        if $|L| = |N_h|$ then
            $h \leftarrow$ get first element of $L$
            place $vm$ to $h$
            return true
        end
    end
    $L \leftarrow$ get all underutilized nodes from $N_h$
    if $L \neq \emptyset$ then
        $L \leftarrow$ get all idle nodes from $N_h$
        if $L \neq \emptyset$ then
            $L \leftarrow$ get all switched-off nodes from $N_h$
            if $L \neq \emptyset$ then
                return false
            end
        end
    end
    sort $L$ in ascending order by proximity
    $h \leftarrow$ get first element of $L$
    place $vm$ to $h$
    return true
end
```

In other words, the VM placement algorithm will avoid bringing an *ok* node into the *overutilized* state, unless no other nodes can accept the VM instance without suffering similar consequence (i.e., themselves becoming overutilized).

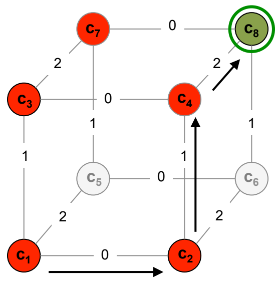

Fig. 3. Illustration of the initial VM placement algorithm in a three-dimensional binary hypercube with five *overutilized*, one *ok*, and two *switched-off* nodes.

Example. Let's demonstrate how the initial VM placement algorithm works. For simplicity and brevity, we consider the 3-dimensional binary hypercube comprising eight compute nodes of Fig. 3. Let's assume that a new VM provisioning request arrives at the data center, and that the random walk within the hypercube that was initiated upon receipt of the request lands on compute node c_1. This node will become the VM initiator. According to the content of its local cache, the only node currently being in the *ok* state is c_8. Hence, c_1 will attempt to place the new VM instance on c_8.

Since c_8 and c_1 are not direct hypercube neighbors, c_1 has to resolve the shortest routing path to c_8. This is readily done by using the unique hypercube identifiers of the two nodes [41]. In our example, it turns out the shortest routing path is $c_1 \rightarrow c_2 \rightarrow c_4 \rightarrow c_8$.

Using this route, c_1 starts the negotiation with c_8, asking for its current power consumption, $p^{c_8}(t)$, and its energy profile, E^{c_8}. Based on this information, c_1 verifies that c_8 has the capacity to provision the new VM instance, and therefore it sends the related package to this compute node. At this point, the initial VM placement algorithm successfully terminates.

3.4 Load Balancing Strategy

At the end of each synchronization period T, each compute node $i = 1..|C_t|$, with $|C_t|$ denoting the current number of active compute nodes, selects among the following actions based on its current power consumption:

- If $p_i(t) \geq p^i_{max}$, the compute node will try to migrate as many VMs as needed in order to reduce its power consumption below the maximum threshold.
- If $p_i(t) \leq p^i_{min}$, the compute node will try to migrate all its hosted VMs in order to switch off.
- Finally, if $p^i_{min} < e_i(t) < p^i_{max}$, the compute node will maintain all its hosted VMs until the end of the next period.

As the last action requires no further explanation we focus on how the first two actions are realized by each compute node.

3.4.1 *Partial Migration*

If a compute node has exceeded its maximum power consumption threshold, it has become overutilized and is likely to violate the SLAs of the VM instances that is currently hosting. To mitigate this risk, the overutilized compute node will start a *partial* VM migration process in order to shift part of its current workload to one or more nodes in the hypercube, whose current status is *ok*, *idle*, *underutilized*, or *switched-off*, so as to eventually reduce its power consumption to an acceptable level. Algorithm 2 sketches this procedure.

The compute node retrieves its local cache of hypercube nodes, N_h, (sorted in descending order according to their current power consumption), and, for each one, it performs the following steps: it first examines the node's current power consumption and power profile. This information is immediately available to the compute node, as it is communicated between the hypercube neighbors through the periodically exchanged hypercube heartbeat messages. Based on this information, the compute node checks if the currently processed node is overutilized. If so, the compute node moves on to the next entry of N_h. Otherwise, the compute node enters a loop in an effort to transfer as many

Algorithm 2: Partial VM migration

```
input: Compute node $c = \{id, W(t), p(t), s(t), N_h, E\}$
begin
    sort $N_h$ in descending order by power consumption
    foreach compute node $h \in N_h$ do
        if $h$ has state $s_h(t) = overutilized$ then
            continue
        end
        while true do
            if $|W(t)| = 0$ or $s(t) \neq overutilized$ then
                return
            end
            $vm \leftarrow$ get next VM instance from $W(t)$
            if $p_{vm} \geq (p^h_{max} - p^h(t))$ then
                continue
            end
            if hwReqMet (h, vm) then
                if $s_h(t) = switched\text{-}off$ then
                    switch on $h$
                end
                migrate $vm$ from $c$ to $h$
            end
        end
    end
end
```

Algorithm 3: Full VM migration

```
input: Compute node $c = \{id, W(t), p(t), s(t), N_h, E\}$
begin
    sort $N_h$ in descending order by power consumption
    foreach compute node $h \in N_h$ do
        if $s_h(t) \in \{switched\text{-}off, overutilized\}$ then
            continue
        end
        while true do
            if $W(t) = \emptyset$ or $s_h(t) = overutilized$ then
                break
            end
            $vm \leftarrow$ get next VM instance from $W(t)$
            if $p_{vm} \geq (p^h_{max} - p^h(t))$ then
                continue
            end
            if hwReqMet (h, vm) then
                migrate $vm$ from $c$ to $h$
            end
        end
        if $W(t) = \emptyset$ then
            switch off compute node $c$
            break
        end
    end
end
```

VM instances as possible. The decision to migrate -or not- a VM instance is determined by the condition of Equation 7.

The above equation ensures that, by accepting VM instances from other nodes, each compute node will not become itself overutilized. Hence, as long as the condition of Equation 7 is not met, and the VM instance's hardware requirements are met, the currently selected VM instance will be migrated to the currently selected node. If the latter is not switched on, the compute node will have to switch it on before beginning the live migration.

The aforementioned process is repeated until (i) all contents of N_h have been processed, (ii) the compute node has adequately reduced its power consumption, or (iii) the compute node's workload becomes empty.

3.4.2 *Full Migration*

At the end of each synchronization period T, and after all overutilized nodes have completed their respective VM migrations, if a compute node is found to be underutilized, it is for the benefit of the data center's overall power consumption to attempt to shift all its workload and subsequently be switched off.

The full migration process of Algorithm 3 resembles the one used for partial VM migration, though there are two differences:

- Switched-off compute nodes are ignored. The reason is, we would not improve the power consumption of the data center if we switched off a compute node at the expense of switching on one or more of its neighbors.
- The algorithm attempts to shift the entire workload (VM instances) of the compute node, so that upon completion, the compute node can shut down.

Concluding, we should indicate that the full migration process does not guarantee that the compute node will always manage to migrate all its VM instances and switch off. The extent of workload shifting primarily depends on the state of other nodes in the hypercube topology. Again, our algorithms do not aim at shutting down compute nodes with the side-effect of overutilization of the data center's remaining resources, as such a strategy would increase the danger of SLA violations, and would negatively contribute to the overall reliability of the data center.

3.5 Workload Management Example

Let's now demonstrate how our workload management approach works: consider a data center that consists of eight homogeneous compute nodes, all organized into a three-dimensional binary hypercube. The compute nodes have a common power profile:

$$p_{idle} = 160\text{W}, p_{min} = 180\text{W}, p_{max} = 250\text{W}$$

As it is shown in the first, left-side part of Fig. 4, currently only five out of the eight compute nodes are active, and are provisioning VM instances which, for the sake of simplicity, consume 5W each. The example illustrates how the distributed load balancing scheme is applied. Our narration is divided in three phases that take place simultaneously and pertain to the VM migration strategies of compute nodes c_1, c_5, and c_6, as follows:

Phase 1: At the end of period T, compute node c_1 is hosting 20 VM instances, has a power consumption of 260W, and is therefore considered *overutilized*. To remedy this, c_1 will attempt to shift part of its workload to one or more of the not overloaded nodes, namely c_2, c_3, c_5, and c_6, until its power consumption drops below the acceptable threshold $p_{max} = 250$W. After retrieving their current state from its local cache, N_h, c_1 will start with the most utilized ones, and will thus first try to migrate VM instances to its *ok* neighbor c_3, which currently has a power consumption of

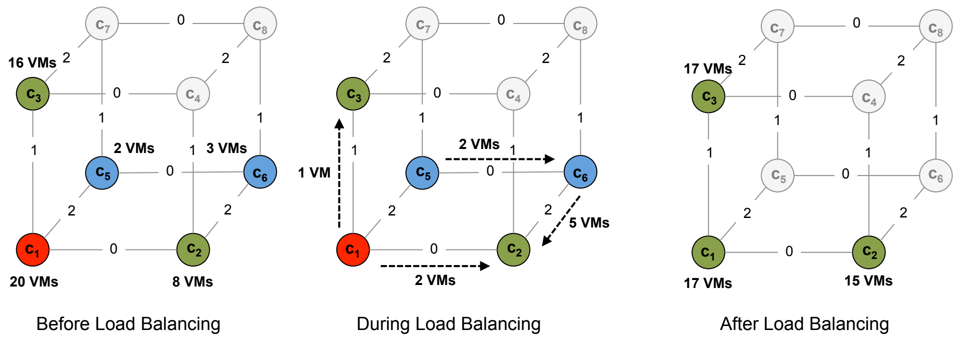

Fig. 4. Illustration of the proposed decentralized load balancing scheme.

Compute node	Before load balancing			After load balancing		
	VM instances	Power consumption	State	VM instances	Power consumption	State
c1	20	260W	overutilized	17	245W	ok
c2	8	200W	ok	15	225W	ok
c3	16	240W	ok	17	245W	ok
c4	0	0W	switched off	0	0W	switched off
c5	2	170W	underutilized	0	0W	switched off
c6	3	175W	underutilized	0	0W	switched off
c7	0	0W	switched off	0	0W	switched off
c8	0	0W	switched off	0	0W	switched off
	49	**1045W**		**49**	**715W**	

Fig. 5. Summary of the data center's status before and after the application of load balancing.

240W. Taking the first VM instance out of its workload, c_1 assesses the condition as expressed in Equation 7, and finds out that it can perform the migration. Continuing with the next VM instance, c_1 reassesses the condition, but this time it turns out that c_3 cannot accept any more workload without moving to the *overutilized* state. Hence, c_1 picks the next neighbor, c_2, and repeats the same process. As the current power consumption of c_2 is 200W, c_1 is allowed to shift vm_2 to it along with the next VM instance, vm_3. At this point, the power consumption of c_1 has dropped to 245W, and its state has become *ok*. Hence, the partial VM migration process successfully terminates.

Phase 2: Now, let us examine compute node c_5, which is underutilized as it is currently hosting two VM instances, and is consuming 170W. According to our load balancing approach, c_5 will try to migrate both VM instances and switch off. Like already described, c_5 first retrieves from its local cache the current state and power consumption of the other nodes. Since all nodes are synchronized, at this time the state of c_1 is still *overutilized*, c_3 and c_2 are *ok*, while the state of neighbor c_6 is *underutilized*. As Fig. 4 shows, there are no other active nodes. Hence, c_5 will migrate its two VM instances to c_2 through c_6 and switch off. The power consumption of c_2 will be increased by 20W.

Phase 3: While the activities of phases 1 and 2 take place, compute node c_6, which is also underutilized and currently hosting three VM instances, will try to shift all its workload to other non-overloaded nodes in order to switch off. At this point, c_6 knows of two *ok* nodes: c_2 and c_3. Starting with the closest one, c_6 will eventually manage to migrate all three VM instances to c_2, eventually raising its power consumption to 225W (remember that, around the same time, c_2 accepts two VM instances from c_1). Now, while migrating its VM instances, c_2 also receives the two VM instances from c_5, which in the meantime has switched off. Since c_2 is still in *ok* state and is able to accept more workload without becoming overutilized, c_6 will also forward those last two VM instances and will finally shut down.

Fig. 5 summarizes the condition of the data center before and after the synchronized, distributed load balancing. As it can be seen also in the right-side part of Fig. 4, at the end of this synchronization period, the load balancer managed to i) effectively remedy the overutilization of compute node c_1, ii) consolidate the workload in three compute nodes, namely c_1, c_2, and c_3, and iii) switch off two underutilized compute nodes (c_5 and c_6) that became idle in the process.

4 Evaluation

This section presents the outcomes of an experimental evaluation that we conducted by benchmarking our proposed approach (referred to as *HyperCube Load Balancer*, abbreviated to *HCLB*, hereinafter) against the *V-MAN* load balancer [30]. We selected *V-MAN* particularly because of the fact that, like our approach, it operates in a decentralized manner based on a peer-to-peer overlay topology, although it is less structured than the hypercube. The goal was to assess the similarities and differences between the two approaches, and identify the potential trade-offs in choosing one approach over the other. In order to evaluate our approach in pragmatic enterprise-scale cloud infrastructures, we carried out a series of experiments with the use of a simulation-based implementation in Java. The experiments aimed at examining the following main aspects: i) *elasticity*: adapting to random workload changes; ii) *eradication of under/over-utilized nodes*: balancing underutilized and overutilized physical machines; iii) *power consumption*: energy costs per hour for the data center.

In our simulator prototype, we specified a data center with maximum capacity of 1024 compute nodes. The nodes were organized in a 10-dimensional hypercube overlay in the case of *HCLB*. while in the case of *V-MAN* each node was initialized with ten neighbors. Without loss of generality,

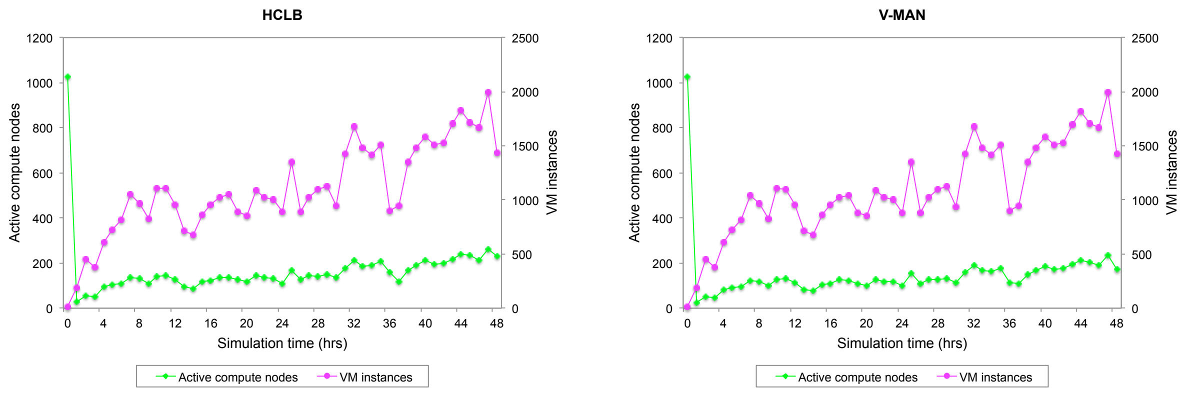

Fig. 6. Number of active compute nodes in the presence of random changes to the overall workload.

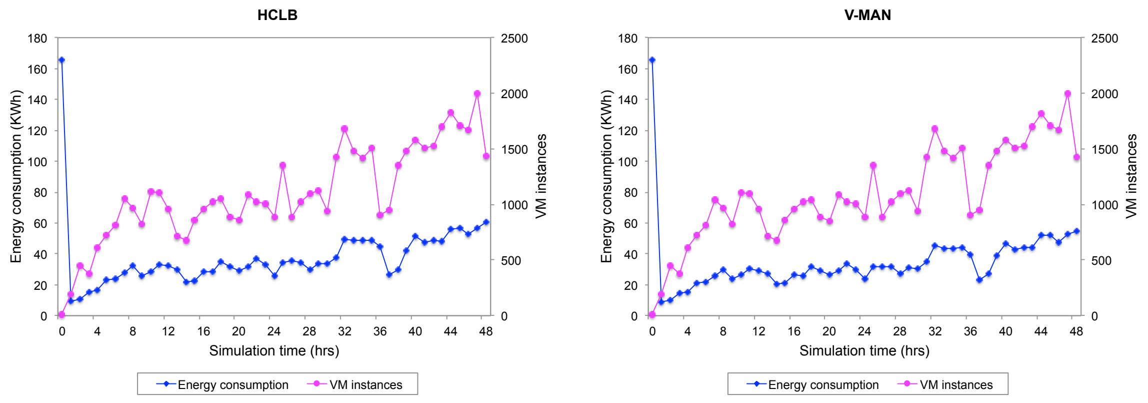

Fig. 7. Overall Energy consumption of the compute nodes in the presence of random changes to the overall workload.

we considered the compute nodes homogeneous in terms of both software and hardware[1], all with the same CPU power profile as follows: power consumption when idle was set at 162 W, the minimum threshold below which the compute node is characterized in our approach as underutilized was set at 180 W, while the maximum power consumption threshold was set at 250 W.

In all conducted experiments, we simulated the operation of the data center over a 48-hour period, with a per-second sampling rate of the utilization and power consumption of the individual compute nodes. We set the load balancing period to 60 seconds, which was also the duration of each simulation round. Each running VM instance contributed 10 W overhead in the CPU power consumption of its compute node host. Besides, as the live migration creates an extra CPU load (it has been shown though that the induced overhead is low [11]), we charged an additional 20 W overhead per individual VM migration, and fixed its duration to 10 seconds. Finally, the cost of switching on/off an individual compute node was set to an average value of 100 W, and the switch on/off duration was also set to a fixed amount of 10 seconds.

1. This is, however, not a hard requirement in our approach, as our migration algorithms show.

4.1 Load Balancing on Random Workload Changes

In the first experiment, we assessed the elasticity attained by the data center thanks to the energy-efficient workload management of both our approach and *V-MAN*. In this context, elasticity amounts to the number of active (i.e. switched on) compute nodes in relation to the current workload. For both cases, we initialized the data center with 1024 idle compute nodes and randomly distributed 16 VM instances among them. Then, we engaged in a process of increasing/decreasing the workload by randomly adding/removing 4 VM instances per second, for the whole 48-hour duration of the experiment.

Fig. 6 illustrates the dynamic behavior and elastic adaptation of the system resources, in response to the aforementioned random workload changes. As it can be seen, the behavior exhibited by the system was similar in both cases: the number of switched on nodes was dramatically reduced after the first simulation round, given the small number of VM instances. From that point on, the number of active nodes at the end of each simulation round was analogous to the number of VM instances.

In both *HCLB* and *V-MAN*, such elastic reaction to workload changes positively affected the energy consumption of the data center, as shown in Fig. 7. Indeed, following the workload trend, the data center's energy consumption was increased and decreased as the compute nodes were switched on and off. Hence, the experiment verified that,

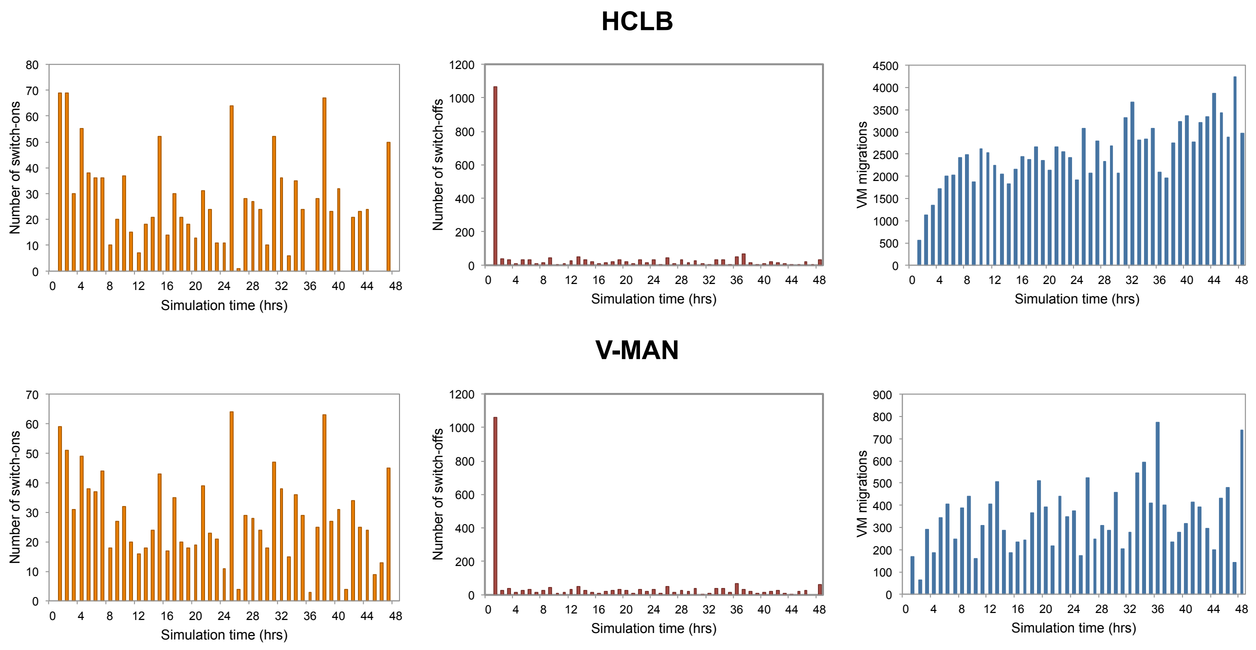

Fig. 8. Background activity (compute node switch-ons, switch-offs, and VM migrations) in the presence of random changes to the overall workload.

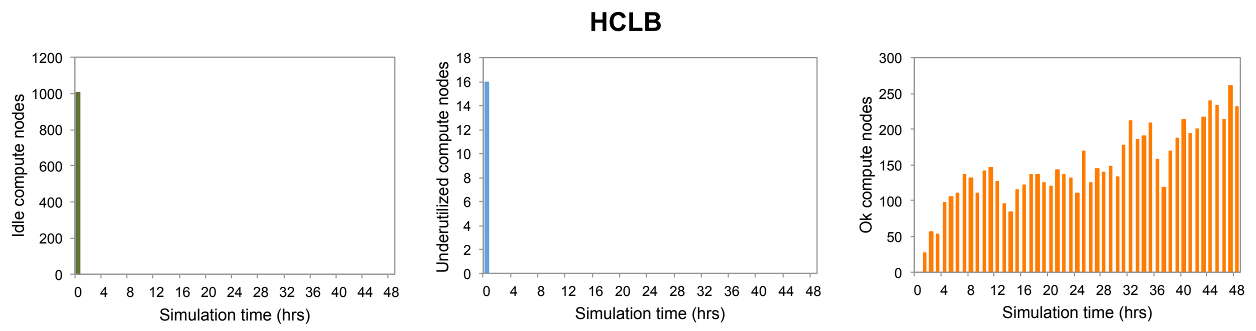

Fig. 9. Changes of state of the active compute nodes in *HCLB*.

thanks to the dynamic and distributed load balancing of our approach, the data center as a whole is able to readjust its overall energy consumption by consolidating the VM instances in as few compute nodes as deemed necessary. Furthermore, the measured energy efficiency was similar to that of *V-MAN*, which was expected to some extent, as in both approaches the compute nodes are able to exploit their “global view” of the overall status of the data center upon load balancing.

Fig. 8 illustrates the background activity of the data center, in terms of the number of VM migrations, as well as the number of compute node switch-ons and switch-offs that were triggered by each load balancer over the course of the simulated 48-hour operation. As it can be seen, *HCLB* and *V-MAN* carried out similar numbers of switch-ons and -offs in order to re-adjust their resources and re-distribute the VM instances, responding to the random changes in the overall workload. Hence, the number of shutdowns of the compute nodes in both cases is explained by the constant effort of the two algorithms to turn off any idle or underutilized compute nodes, whenever those appear. Still, the number of VM migrations was higher in the case of *HCLB*. As the measurements demonstrated, *HCLB* chose to switch on additional compute nodes instead of overloading the already active ones, whenever an increase in the workload occurred, whereas *V-MAN* tended to use as few compute nodes as possible, without paying attention to their potential overuse.

The above finding is clearly manifested in Fig. 9 and Fig. 10, which show the states of the active compute nodes in *HCLB* and *V-MAN*, respectively, throughout the experiment. In the *HCLB* case, the vast majority of the active compute nodes was in *ok* state, meaning they operated within acceptable power range, not being underutilized or overutilized. Due to our hypercube-based load balancing scheme, all underutilized nodes were easily detected at the end of each simulation round, and were effectively switched off after migrating their VM instances to the available *ok* nodes. On the other side, *V-MAN* maintained a considerable number of active compute nodes in the *overutilized* state, instead of trying to reduce their workload by migrating some of their VM instances to other nodes. While such decision

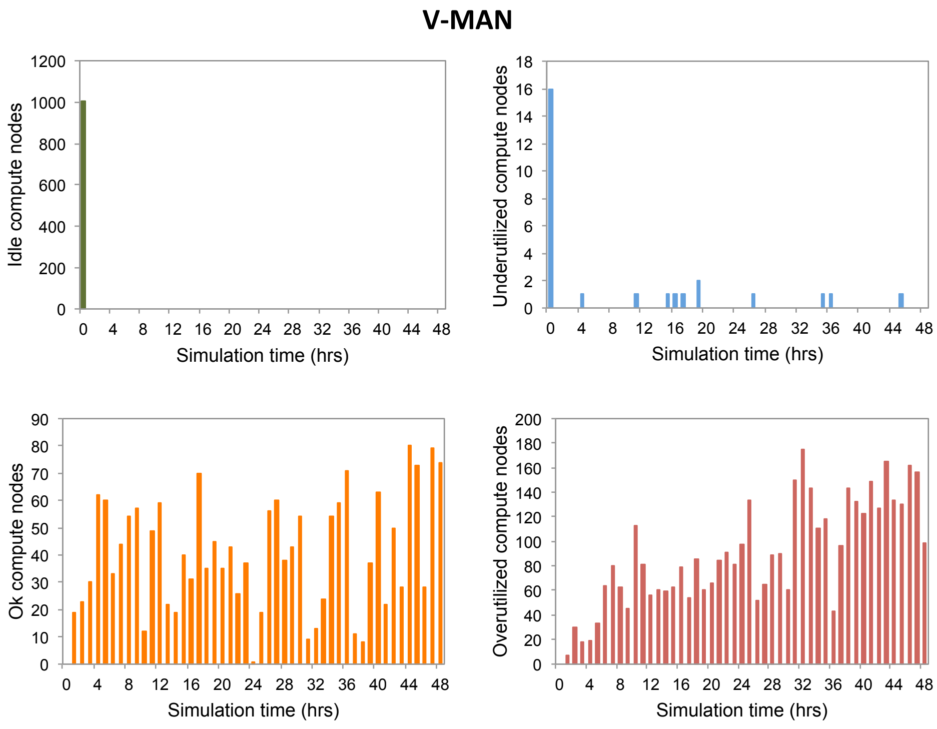

Fig. 10. Changes of state of the active compute nodes in *V-MAN*.

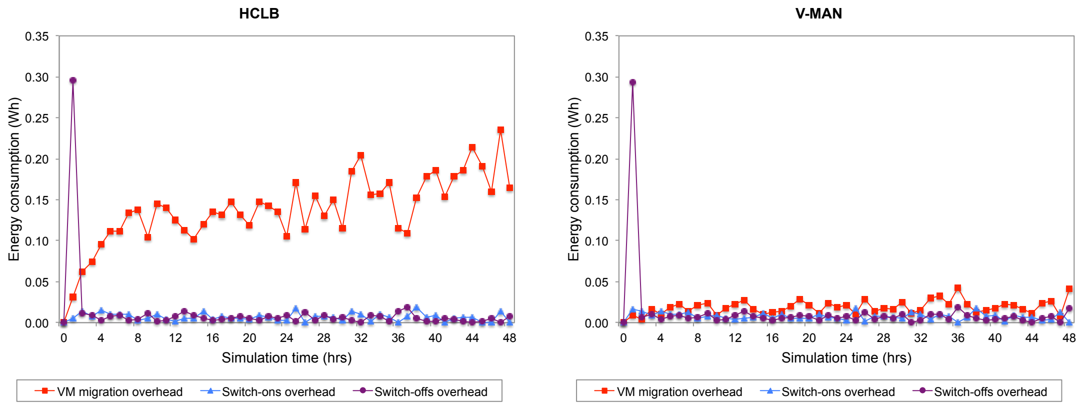

Fig. 11. Energy consumption overheads in the presence of random changes to the overall workload.

produced some minor benefits in terms of reduced energy consumption, it is safe to argue that it is also likely to yield SLA violations in the long term, as the performance of the overutilized nodes will be gradually deteriorated.

Finally, the key difference in the behavior of the two approaches is also reflected in the measured energy consumption overheads that were inflicted by the background activity of the data center (i.e., the VM migrations and the switching on/off of compute nodes) during the experiment, as shown in Fig. 11. Nevertheless, compared to the energy consumption of the active compute nodes, the measurements suggest that those overheads are not significant, although in a real-world setting there may be more variations in the time needed for a switch off/on, or for a live VM migration.

4.2 Load Balancing on Increasing Workload

In the second experiment, we performed a stress test on the data center. Specifically, we initialized the data center with 1024 compute nodes in idle state, and then started *just* adding workload with a rate of 2 VM instances per minute, for the entire 48-hour period. The goal was to measure the energy consumption of the data center in the two cases of using *HCLB* and *V-MAN*, as well as to observe the utilization of the data center's compute nodes in response to the progressively increasing workload.

The left-side part of Fig. 12 illustrates the energy consumption measurements. Apparently there was no significant difference between *HCLB* and *V-MAN*, while the overall consumption was at all times analogous to the current workload. At the end of the first hour of the experiment, the data center had employed 15 active compute nodes

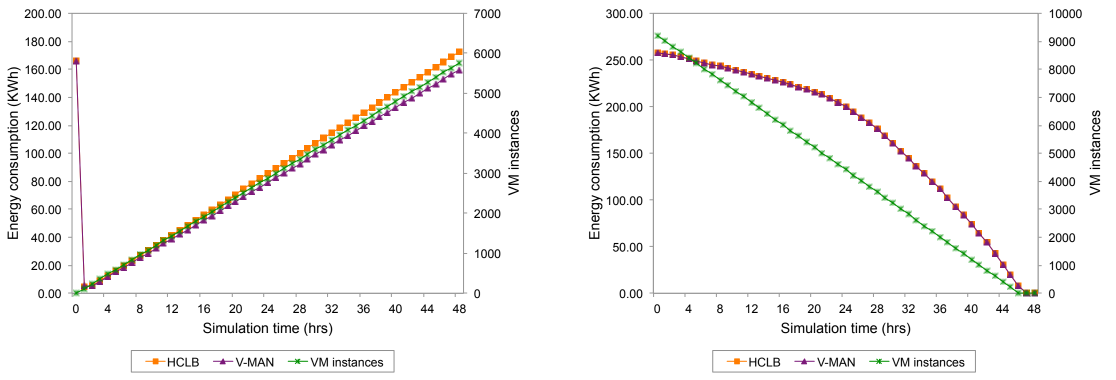

Fig. 12. Comparison of energy consumption while balancing a continuously increasing workload.

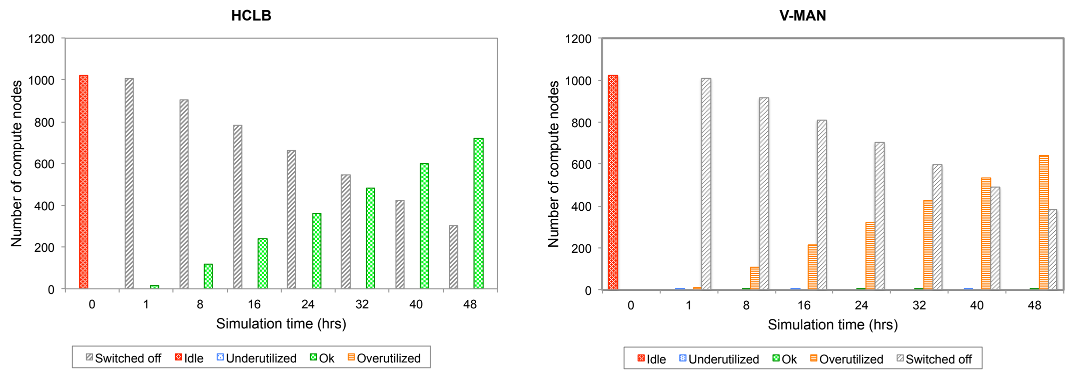

Fig. 13. Changes in the utilization and state of compute nodes in order to balance a continuously increasing workload.

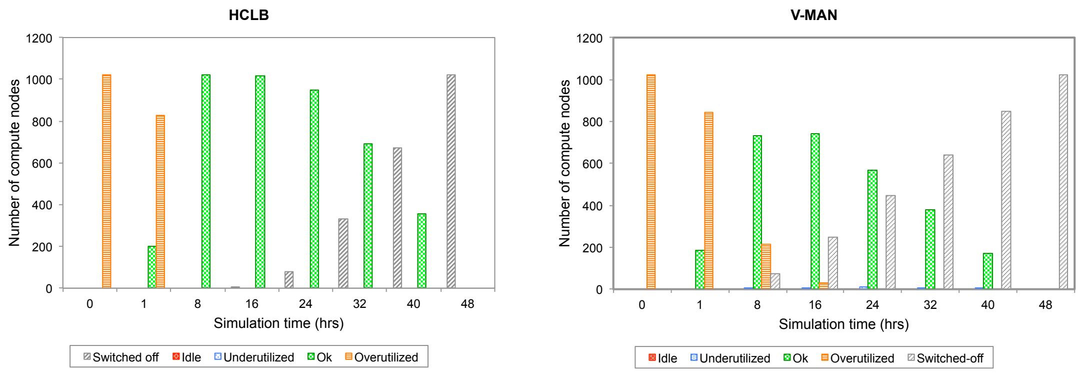

Fig. 14. Changes in the utilization and state of compute nodes in order to balance a continuously decreasing workload.

out of the 1024 available compute nodes, switching off the remaining ones in order to reduce its overall energy consumption. Following the same behavioral pattern, as the workload increased, more compute nodes where turned on and utilized to provision the added VM instances. At the end of the experiment, more than half of the available compute nodes had effectively become active, while the data center still managed to reserve 290 switched off compute nodes.

Still, the two approaches managed their compute nodes in a different manner. As it can be seen in Fig. 13, *HCLB* exclusively used compute nodes in the *ok* state, with the load balancing algorithms making sure no compute node was overloaded, even if that led to switching on additional nodes. On the other hand, the *V-MAN* load balancer packed as many VM instances as possible to the available compute nodes, eventually managing to switch off more nodes than *HCLB*, though at the expense of bringing the active ones into the *overutilized* state. Again, as it was evidenced in the previous experiment, the energy benefits of *V-MAN* compared to *HCLB* were limited, while the risk of violating SLAs due to reduced performance of the overutilized compute nodes was naturally increased.

4.3 Load Balancing on Decreasing Workload

The third and final experiment was a reverse take of the previous stress test, where we assessed the behaviour of the data center in response to a progressively decreasing workload. The goal was to evaluate the ability of the *HCLB* and *V-MAN* load balancing mechanisms to quickly switch off any unnecessary resources in order to reduce the energy footprint of the data center. More specifically, we initialized the data center with 1024 overutilized compute nodes, and then started reducing the number of VM instances at a rate of 200 per hour.

As Fig. 12 (the right-side part) and Fig. 14 show, both approaches exhibited an almost identical behavior in reducing the energy consumption, as in this case both *HCLB* and *V-MAN* started switching off the unnecessary compute nodes. Still, it is worth noting that, the two approaches again managed the active compute nodes in a different manner. In addition to switching off any underutilized or idle compute nodes, and whenever there was a chance to do so, *HCLB* shifted part of the workload from overutilized nodes to the ones operating at the *ok* state. This way, *HCLB* managed to eliminate all overutilized compute nodes from the data center much sooner than *V-MAN*.

Throughout the conducted experiments, it became clear that, although the energy footprint generated by the two approaches is comparable, *V-MAN* tends to overutilize its nodes. Over time, such behavior will most likely lead to SLA violations. In contrast, *HCLB* systematically avoids pushing its compute nodes into the *overutilized* state, refraining so from such load balancing-induced violations. Moreover, *HCLB* offers a comprehensive set of protocols that allow for VM instances to gracefully join and depart the cloud infrastructure. As such, we consider *HCLB* as a more appropriate choice for enterprise-grade data centers where the cloud service provider reputation always remains at stake.

5 Conclusion

We presented a fully decentralized approach for managing the workload of large, enterprise cloud data centers in an energy-efficient manner. Our approach comprises a hypercube overlay for the organization of the data center's compute nodes, and a set of distributed load balancing algorithms, which rely on live VM migration to shift workload between nodes, with the dual goal to i) minimize the active resources of the data center, and thereby its energy consumption, and ii) avoid overloading of compute nodes.

We conducted a series of simulation-based experiments in order to evaluate our proposed approach. The results suggest that our decentralized load balancer is scalable, as it operates in a similar way regardless of the data center size, and energy-efficient. Moreover, it enables automated elasticity as the data center's compute nodes are switched on and off on demand, in response to the changes in the data center's overall workload. Our experimental results also showed that the cummulative cost of live VM migrations along with that of switching on and off compute nodes is insignificant compared to the energy savings attained by our approach.

In future work, we plan to implement and integrate our decentralized wokload manager in an open-source cloud operating system, such as e.g., OpenStack [36] or OpenNebula [42]. Such implementation will also allows us to experiment with real-world use cases, although such experimentation is likely to be carried out at a smaller scale than our simulations, due to lack of access to large-scale physical resources. Furthermore, we would like to expand our model in order to consider the power consumptions inflicted by other resources of the data center, such as the compute nodes disk, RAM memory, and network, even though those are usually overruled by the CPU power consumption. Along the same line, we plan to investigate ways to introduce additional parameters into our load-balancing algorithms, so as accommodate less homogeneous settings where, on the one hand, the compute nodes offer different hardware and/or software resources, while the VM instances pose different hardware requirements too. Communication-aware VM scheduling approaches such as the one proposed by Guan et al. [43] could also be effectively combined with our load-balancing scheme to allow for a more fine-grained selection of the VM instances to be migrated. Finally, we are interested in incorporating appropriate VM and VM migration power–metering techniques and mechanisms [44] [45] that will allow us to assess the efficacy of our approach on the basis of more accurate and pragmatic power metrics.

Acknowledgments

We sincerely thank the reviewers for their comments and suggestions that help us improve our work. This research has been supported by a *THALES* grant co-financed by EU-ESF and the Greek NSFR, the *SUCRE EU-FP7* project, and the *Onassis Foundation*.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[3] L. A. Barroso and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-scale Machines*, 1st ed. Morgan and Claypool Publishers, 2009.

[4] ——, "The Case for Energy-proportional Computing," *IEEE Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[5] A. Kulkarni, "Dynamic Virtual Machine Consolidation," Google Patents, September 5th, 2012, US Patent App. 13/604,134.

[6] A. Roytman, A. Kansal, S. Govindan, J. Liu, and S. Nath, "PAC-Man: Performance Aware Virtual Machine Consolidation," in *Proc. of 10th Int. Conf. on Autonomic Computing (ICAC'13)*. San Jose, CA: USENIX Association, June 2013, pp. 83–94.

[7] T. C. Ferreto, M. A. Netto, R. N. Calheiros, and C. A. De Rose, "Server Consolidation with Migration Control for Virtualized Data Centers," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1027–1034, 2011.

[8] W. Shi and B. Hong, "Towards Profitable Virtual Machine Placement in the Data Center," in *Proc. of the 4th IEEE Int. Conf. on Utility and Cloud Computing (UCC'11)*, Melbourne, Australia, December 2011, pp. 138–145.

[9] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *Proc. of 2nd Symposium on Networked Systems Design & Implementation-Volume 2 (NSDI'05)*. Oakland, CA: USENIX Association, May 2005, pp. 273–286.

[10] K. Tsakalozos, V. Verroios, M. Roussopoulos, and A. Delis, "Time-Constrained Live VM Migration in Share-Nothing IaaS-Clouds," in *Proc. of the 7th IEEE Int. Conf. on Cloud Computing (IEEE CLOUD'14)*, Anchorage, AK, June 2014.

[11] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation," in *Proc. of the 1st Int. Conf. on Cloud Computing (CloudCom'09)*, Beijing, China, December 2009, pp. 254–265.

[12] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi, "Reactive Consolidation of Virtual Machines Enabled by Postcopy Live Migration," in *Proc. of 5th Int. Workshop on Virtualization Technologies in Distributed Computing (VTDC'11)*, San Jose, CA, June 2011, pp. 11–18.

[13] M. H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya, "Virtual machine consolidation in cloud data centers using aco metaheuristic," in *Euro-Par 2014 Parallel Processing, Lecture Notes in Computer Science Volume 8632*. Springer, 2014, pp. 306–317.

[14] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a Consolidation Manager for Clusters," in *Proc. of 2009 ACM SIGPLAN/SIGOPS Int. Conf. on Virtual Execution Environments (VEE'09)*, Washington, DC, March 2009, pp. 41–50.

[15] E. Feller, L. Rilling, and C. Morin, "Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds," in *Proc. of 12th IEEE/ACM Int. Symposium on Cluster, Cloud and Grid Computing (CCGRID'12)*, Ottawa, Canada, May 2012, pp. 482–489.

[16] F. Quesnel, A. Lèbre, and M. Südholt, "Cooperative and Reactive Scheduling in Large-scale Virtualized Platforms with DVMS," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1643–1655, 2013.

[17] J. Baliga, R. W. Ayre, K. Hinton, and R. Tucker, "Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149–167, 2011.

[18] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, and K. Pentikousis, "Energy-Efficient Cloud Computing," *The Computer journal*, vol. 53, no. 7, pp. 1045–1051, 2010.

[19] A. M. Sampaio and J. G. Barbosa, "Optimizing Energy-Efficiency in High-Available Scientific Cloud Environments," in *Proc. of 3rd Int. Conf. on Cloud and Green Computing (CGC'13)*, Karlruhe, Germany, September 2013, pp. 76–83.

[20] C. Mastroianni, M. Meo, and G. Papuzzo, "Probabilistic Consolidation of Virtual Machines in Self-Organizing Cloud Data Centers," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 215–228, 2013.

[21] A. Beloglazov and R. Buyya, "Energy Efficient Resource Management in Virtualized Cloud Data Centers," in *Proc. of the 10th IEEE/ACM Int. Conf. on Cluster, Cloud and Grid Computing (CCGRAID'10)*, Melbourne, Australia, May 2010, pp. 826–831.

[22] 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*, vol. 28, no. 5, pp. 755–768, 2012.

[23] B. Dougherty, J. White, and D. C. Schmidt, "Model-driven Auto-Scaling of Green Cloud Computing Infrastructure," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 371–378, 2012.

[24] D. Bruneo, A. Lhoas, F. Longo, and A. Puliafito, "Analytical Evaluation of Resource Allocation Policies in Green IaaS Clouds," in *Proc. of the 3rd Int. Conf. on Cloud and Green Computing (CGC'133)*, Karlruhe, Germany, September 2013, pp. 84–91.

[25] Y. Lee and A. Zomaya, "Energy Efficient Utilization of Resources in Cloud Computing Systems," *The Journal of Supercomputing*, vol. 60, no. 2, pp. 268–280, 2012.

[26] J. Cao, Y. Wu, and M. Li, "Energy Efficient Allocation of Virtual Machines in Cloud Computing Environments Based on Demand Forecast," in *Proc. of 7th Int. Conf. on Grid and Pervasive Computing (GPC'12), Lecture Notes in Computer Science Volume 7296*. Springer, 2012, pp. 137–151.

[27] M. Mazzucco, D. Dyachuk, and R. Deters, "Maximizing Cloud Providers' Revenues via Energy Aware Allocation Policies," in *Proc. of the 3rd IEEE Int. Conf. on Cloud Computing (CLOUD 2010)*, Miami, FL, May 2010, pp. 131–138.

[28] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-source Cloud-Computing System," in *9th IEEE/ACM Int. Symposium on Cluster Computing and the Grid (CCGRID'09)*, Shanghai, China, 2009, pp. 124–131.

[29] P. Graubner, M. Schmidt, and B. Freisleben, "Energy-Efficient Management of Virtual Machines in Eucalyptus," in *Proc. of 2011 IEEE Int. Conf. on Cloud Computing (CLOUD'11)*, Washington, DC, July 2011, pp. 243–250.

[30] M. Marzolla, O. Babaoglu, and F. Panzieri, "Server Consolidation in Clouds Through Gossiping," in *Proc. of 2011 IEEE Int. Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'11)*, Lucca, Italy, June 2011, pp. 1–6.

[31] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. Van Steen, "Gossip-based Peer Sampling," *ACM Transactions on Computer Systems (TOCS)*, vol. 25, no. 3, p. 8, 2007.

[32] D. Barbagallo, E. Di Nitto, D. J. Dubois, and R. Mirandola, "A Bio-inspired Algorithm for Energy Optimization in a Self-organizing Data Center," in *Proc. of the 1st Int. Conf. on Self-organizing Architectures (SOAR'09)*. Cambridge, United Kingdom: Lecture Notes in Computer Science 6090, Springer-Verlag, 2010, pp. 127–151.

[33] J. L. Hellerstein, "Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 14–28, 2014.

[34] W. Fang, X. Liang, S. Li, L. Chiaraviglio, and N. Xiong, "VM-Planner: Optimizing Virtual Machine Placement and Traffic Flow Routing to Reduce Network Power Costs in Cloud Data Centers," *Computer Networks*, vol. 57, no. 1, pp. 179–196, 2013.

[35] V. Mann, A. Kumar, P. Dutta, and S. Kalyanaraman, "VMFlow: Leveraging VM Mobility to Reduce Network Power Costs in Data Centers," in *Proc. of the 10th Int. IFIP TC 6 Conf. on Networking - Volume Part I (NETWORKING'11)*. Valencia, Spain: Springer-Verlag, 2011, pp. 198–211.

[36] K. Pepple, *Deploying OpenStack*. Sebastopol, CA: O'Reilly Media, Inc., 2011.

[37] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl, "HyperCuP – Hypercubes, Ontologies, and Efficient Search on Peer-to-peer Networks," in *Agents and Peer-to-Peer Computing*. Lecture Notes in Computer Science Volume 2530, Springer, 2003, pp. 112–124.

[38] M. T. Schlosser, "Semantic web services," Master's thesis, Universität Hannover, Institut für Mikroelektronische Schaltungen und Systeme, 2002.

[39] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[40] C. A. Waldspurger, "Memory Resource Management in VMware ESX Server," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 181–194, 2002.

[41] M. Schlosser, "Semantic Web Services," Master's thesis, Universitt Hannover, 12 2002.

[42] D. Milojičić, I. M. Llorente, and R. S. Montero, "Opennebula: A Cloud Management Tool," *IEEE Internet Computing*, vol. 15, no. 2, pp. 11–14, 2011.

[43] B. Guan, J. Wu, Y. Wang, and S. U. Khan, "CIVSched: A Communication-Aware Inter-VM Scheduling Technique for Decreased Network Latency between Co-Located VMs," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 320–332, 2014.

[44] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual Machine Power Metering and Provisioning," in *Proc. of 1st ACM symposium on Cloud Computing (SOCC'10)*, Indianapolis, IN, June 2010, pp. 39–50.

[45] Q. Huang, F. Gao, R. Wang, and Z. Qi, "Power Consumption of Virtual Machine Live Migration in Clouds," in *Proc. of 3rd Int. Conf. on Communications and Mobile Computing (CMC'11)*, Qingdao, China, April 2011, pp. 122–125.

Michael Pantazoglou is a research associate at the Department of Informatics and Telecommunications of the University of Athens, Greece. His current research interests span the areas of cloud computing, service-oriented computing, with a focus on service interoperability, discovery, and composition, and P2P computing. He holds a PhD in Computer Science from the Univ. of Athens.

Gavriil Tzortzakis is an MSc student at the Department of Informatics and Telecommunications of the University of Athens. His research interests are in cloud computing and wireless sensor networks. He holds a BSc from the Univ. of Athens and is currently a member of the MaDgIK Research Group (http://www.madgik.di.uoa.gr).

Alex Delis is a Professor of Computer Science and a member of the MaDgIK Research Group (http://www.madgik.di.uoa.gr) at the University of Athens. His interests are in distributed computing, virtualized infrastructures and data management and his work has been supported by agencies and organizations in the EU, USA, and Australia. He received both his PhD and MS in Computer Science from the Univ. of Maryland at College Park and holds a Diploma in Computer Engineering from the Univ. of Patras.