Resource optimization across the cloud stack

Published in IEEE Transactions on Parallel and Distributed Systems, 29(1):169-182, 2018

Zoltán Ádám Mann

Abstract—Previous work on optimizing resource provisioning in virtualized environments focused either on mapping virtual machines (VMs) to physical machines (PMs) or mapping application components to VMs. In this paper, we argue that these two optimization problems influence each other significantly and in a highly non-trivial way. We define a sophisticated problem formulation for the joint optimization of the two mappings, taking into account sizing aspects, colocation constraints, license costs, and hardware affinity relations. As demonstrated by the empirical evaluation on a real-world workload trace, the combined optimization leads to significantly better overall results than considering the two problems in isolation.

Index Terms—Virtual machines; VM placement; VM consolidation; VM selection; VM sizing; cloud computing; data center

1 Introduction

As cloud data centers (DCs) are serving an ever growing demand for computation, storage, and networking, their efficient operation has become a high priority. Cloud providers seek to serve as many customer requests as possible and to decrease operational costs. Operational costs are largely driven by electricity consumption, which also impacts the environment. At the same time, cloud providers must also fulfill service-level objectives (SLOs) on performance, availability, and security.

Virtualization has been widely adopted in DCs to consolidate workload on the necessary number of physical machines (PMs) with high utilization of the available hardware resources. For this purpose, virtual machines (VMs) are used as the virtual infrastructure for running the workload, enabling the isolated execution of multiple applications on the same PM. However, virtualization also has some drawbacks (e.g., overhead [50]) and limitations (e.g., no perfect isolation of colocated VMs from each other [7], [27]).

Because of its impact on costs, application performance, SLOs, and the environment, optimization relating to the management of VMs has received considerable attention in the last couple of years. As shown in our recent survey [30], most previous research efforts fall into one of two categories: VM placement and VM selection. *VM placement* is a problem faced by Infrastructure-as-a-Service (IaaS) providers: how to determine a mapping of VMs to PMs with the main objective of minimizing overall energy consumption. On the other hand, *VM selection* is faced by IaaS tenants concerned with assigning application components[1] to VMs.

The two problems are quite different: VM placement is about physical resources, their utilization and power consumption, whereas VM selection is concerned with lease costs and application-level performance metrics. The central notion that connects the two perspectives is the VM.

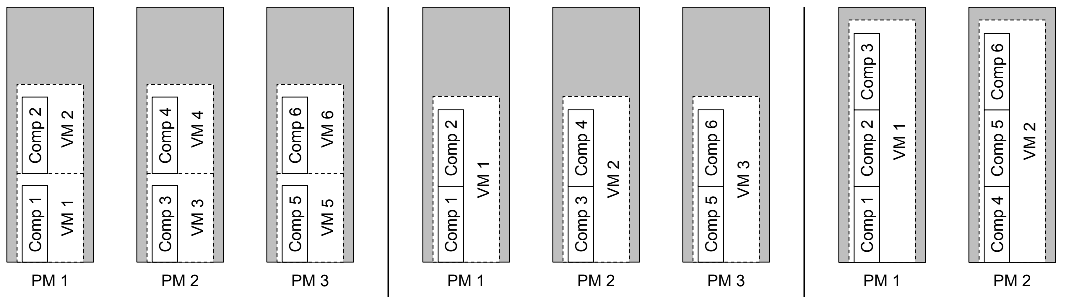
Although VMs play an important role, especially in a public IaaS setting, we argue that *VMs are just a tool for mapping tenants' application components to the provider's PMs* in a safe and manageable fashion. Tenants' main objective is to find hosts for their applications, providers' objective is to utilize their infrastructure by accommodating workload that is valuable for their clients, and thus realize revenue. VMs can be seen as wrappers around application components that make all this possible in a manageable way. In this respect, VM placement and VM selection are just two sides of the same coin. Most importantly, the two problems influence each other.

A simplified example is shown in Fig. 1. Here, we consider a single resource dimension (e.g., only CPU) and assume that all PMs have the same capacity according to this resource. The capacity of the PMs is taken to be 1. We consider six components with resource need 0.3 each (i.e., each component requires 30% of the capacity of a PM). Further, we assume that a VM adds an overhead of 0.05 to the size of the contained component(s) in terms of resource consumption. The three subfigures show the effect of different VM selection policies on VM placement. In Fig. 1(a), the VM selection policy selects a dedicated VM for each component, resulting in 6 VMs of size 0.35 each, the placement of which requires at least 3 PMs. In Fig. 1(b), components are grouped pairwise into VMs, resulting in 3 VMs of size 0.65 each, the placement of which again requires 3 PMs. In Fig. 1(c), groups of 3 components are mapped to VMs, resulting in 2 VMs of size 0.95 each, and these can be hosted by 2 PMs. Therefore, this third scenario leads to approximately 33% energy savings. However, if we continue this line of thought and map 4 components into a single VM, this would result in VMs of size 1.25, which cannot be accommodated by the available PMs without severe resource overload.

As demonstrated by this example, VM selection influ-

• *The author is with the University of Duisburg-Essen, Essen, Germany*

1. In this paper, the term "component" denotes a component of an application, i.e., a software component.



(a) Separate VM for each component (b) 2 components per VM (c) 3 components per VM

Fig. 1. Examples of the impact of VM selection decisions on the possibilities of VM placement

ences VM placement in a non-trivial way. Therefore we argue that, at least in a private cloud setting, where VM selection and VM placement are in the hands of the same organization, the two kinds of optimization should be carried out in a closely coupled way. So, the setting of this paper is the IT department of an organization running a private cloud, in which both VM selection and VM placement are carried out by the IT department. The main questions addressed by this paper are:

- How much can we gain by optimizing VM selection and VM placement together, in a joint optimization?
- If the two problems are solved separately, how much can we gain by incorporating knowledge about VM placement into VM selection and vice versa, by incorporating knowledge about VM selection into VM placement?

To answer these questions, we compare several algorithms, from an integrated approach for the joint selection-and-placement problem to complete separation of the two problems. Also some approaches are investigated that are in between, meaning that they solve the two problems separately but include some information about one of the problems into the solution of the other. To compare the algorithms, we use several metrics including energy consumption, license costs, compliance with hardware affinity constraints, and compliance with colocation constraints of the resulting system configuration.

Next, related work is reviewed in Section 2, followed by the problem formalization in Section 3 and different possible algorithms in Section 4. Empirical experience with applying the presented algorithms to real-world workload data is presented in Section 5 and Section 6 concludes the paper. The online supplemental material contains a more detailed description of the aspects that VM selection and placement need to account for.

2 Previous work

As shown in our recent survey [30], most previous research efforts on VM mapping problems fall into one of two categories: VM placement is concerned with mapping VMs to PMs in a DC, while VM selection considers the problem of mapping application components to VMs. In the taxonomy introduced in [32], the first one is the Single-DC problem, while the latter is the Multi-IaaS problem (mirroring the fact that cloud users can choose among a number of IaaS offerings).

2.1 VM placement

Even within the Single-DC problem, many different problem variants have been considered. The most important differentiating factors are:

- The set of resource types considered:
  - Many papers consider only the CPU [4]–[6], [9], [11], [20], [24].
  - Other papers included, beside the CPU, also some other resources like memory, I/O, storage, or network bandwidth [3], [8], [18], [34], [44].
- The considered cost factors:
  - Many papers focus on the number of active PMs because it largely determines the total energy consumption [4], [11], [39].
  - Some also take into account the load-dependent dynamic power consumption of PMs [1], [5], [15], [18], [20], [43], [47].
  - A further objective of some papers is to minimize the number of overloaded PMs because of the performance degradation that results from overloads [5], [11], [44].
  - Some papers also considered the cost of migration of VMs [5], [11], [37], [43].

As noticed by several researchers, the special case of the Single-DC problem in which a single resource type is considered and the only objective is to minimize the number of used PMs is equivalent to the well-known bin-packing problem. On one hand, this means that the Single-DC problem is strongly NP-hard so that the existence of an efficient exact algorithm is very unlikely. On the other hand, simple packing heuristics like First-Fit (FF), Best-Fit (BF), and First-Fit-Decreasing (FFD) are known to perform very well on bin-packing. Hence, several papers proposed to adopt such heuristics to the VM placement problem [4], [5], [20], [28], [44].

2.2 VM selection

Concerning VM selection (the Multi-IaaS problem), also many different problem formulations have been suggested.

Similarly to the Single-DC problem, most works focus on computational power [12], [29], [46] but a few works also consider other resource types like memory [25], [26], [35]. The main optimization objective is to find the best trade-off between performance and VM lease costs, which typically means that either the minimum required performance is given and costs must be minimized or the acceptable costs are constrained and performance must be maximized. Performance is often defined in terms of the makespan, i.e., the time between starting the first task and finishing the last one, in some cases also allowing dependencies among the tasks [10], [19], [21], [35].

Several different models have been investigated also in terms of VM lease costs. Most works consider costs proportional to VM usage time [9], [12], [22], [29], [45], [46], but some also add fees depending on consumed resource usage [26], [35] or discounts for long-term VM rental [19], [26]. Spot instances have also been considered [14].

Another relevant topic is auto-scaling, aiming to determine the number of necessary instances of a given VM to serve the current load [2], [40]. This can also be seen as a kind of VM selection problem.

2.3 Interplay of VM placement and VM selection

The papers cited above address either VM placement or VM selection in isolation. Although both problems have received much attention, their inter-dependence has hardly been studied. We are aware of only two papers by other researchers that made first steps into this direction. One of them is the recent work of Piraghaj et al. [36]. The focus of that paper is on selecting optimal VM sizes based on the characteristics of the tasks to be allocated. The objective is to reduce energy consumption by minimizing resource wastage. Each VM is assumed to have a fixed size irrespective of its workload, and the difference between the VM's size and the total size of its workload is wasted.

In contrast, this paper assumes that a VM's real size (as taken into account by the provider in VM placement decisions) follows the capacity requirements of its workload. The rationale is that resource usage is most of the time significantly below the peak, yielding a great opportunity for DC operators to consolidate VMs based on their current load and continuously adapt the placement accordingly, always using just the necessary number of active PMs [43]. Another important difference is that the work of Piraghaj et al. [36] did not consider migrations, whereas we do. Through these differences we believe to have a more realistic model, in which the sought trade-offs and the objectives are also somewhat different (opportunities for consolidation through migration versus minimization of wastage through sizing).

The other relevant paper is due to Ganesan et al. [17]. That work is in the context of a Software-as-a-Service provider that wants to allocate the components of its applications to VMs. The focus of the work is on VM sizing, namely, determining the dedicated and shared capacity for the VMs, based on past observations of the applications' workload. Their algorithm also outputs *recommendations* for VM placement, like which VMs can be placed statically and which ones need dynamic placement. However, the actual allocation of VMs to PMs is not carried out; they assume that it is done by some external algorithm. In contrast, we are interested in the impact of selection on placement; it is unfortunately not possible to tell how good that approach is in this respect. Another limitation of that paper is the assumption that each application component is mapped to a separate VM, whereas we also allow to co-locate multiple components in the same VM.

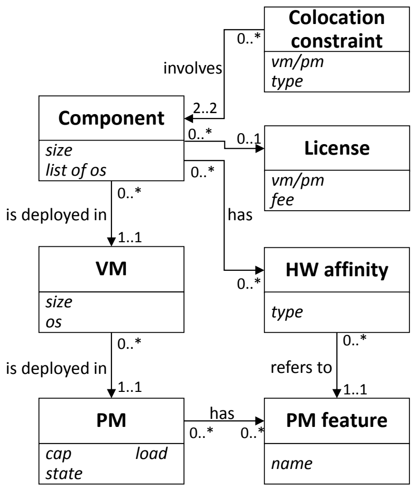


Fig. 2. Problem model using UML notation

In our own previous work, we have started investigating the connections between VM selection and VM placement [33]. In particular, we compared three different VM selection algorithms in combination with the same VM placement algorithm; our results suggested that the more information the VM selection algorithm has about the PMs, the current VM placement, and the VM placement algorithm, the better overall results can be achieved. In that work, components and VMs were only characterized by their size; in contrast, this work analyzes a similar question in the context of a much more general problem formulation, featuring beyond the mere size of the components also license costs, colocation constraints and hardware affinity constraints. Therefore we believe that the results of this paper are more relevant for practical use.

3 Problem Formulation

Based on the analysis of the relevant aspects of the problem (details can be found in the online supplemental material), we came to the problem model summarized in Figure 2.

The problem model revolves around components that are deployed in VMs, which in turn are deployed in PMs.[2] Let C, V, and P denote the set of components, VMs, and PMs, respectively. For a component $c \in C$, $v(c)$ denotes the VM where c is deployed; likewise for a VM $v \in V$, $p(v)$ denotes its hosting PM.

The *size* of a component encodes its resource requirements along multiple resource types as a d-dimensional

2. The numbers near the ends of links mean minimum and maximum cardinalities and * means infinity. For instance, the numbers near the link between Component and VM mean that a component is deployed in exactly one VM, whereas a VM can host an arbitrary number (from 0 to infinity) of components.

vector. Here, d is the number of considered resource types, e.g., if CPU and memory are considered, then $d = 2$. The size of a VM is also a d-dimensional vector: the sum of the sizes of the components deployed in the given VM, plus the overhead of virtualization (the size vector of an empty VM). For a VM $v \in V$, its size thus is computed as

$$size(v) = s_0 + \sum_{c \in C: v(c)=v} size(c),$$

where $s_0 \in \mathbb{R}^d_+$ is the size vector of an empty VM.

Each PM $p \in P$ has given capacity according to each of the considered resource types. Therefore, the capacity of a PM p is given by a d-dimensional vector $cap(p)$. The mapping of VMs on PMs must respect the capacity of the PMs, as captured by the following constraint:

$$\forall p \in P: \quad load(p) = \sum_{v \in V: p(v)=p} size(v) \leq cap(p).$$

Note that here, "$\leq$" is a component-wise comparison of d-dimensional vectors: for $x, y \in \mathbb{R}^d$, $x \leq y$ if and only if $x_j \leq y_j$ for each $j = 1, \ldots, d$.

The *state* of a PM can be either *on* or *off*. The operating system of VM v is $os(v)$. For each component c, the list of operating systems is given on which it can run; $os(v(c))$ must be an element of this list.

A colocation constraint relates to a pair of components. The *type* of the constraint can be one of *must*, *should*, *should not*, and *must not*. Moreover, it is given for each colocation constraint whether it relates to the colocation in the same VM or the same PM. With this mechanism, we can model all colocation aspects described in the online supplemental material. For instance, shared-memory communication between components leads to a *must*-constraint on VM level, meaning that they must be in the same VM, whereas intensive but loosely-coupled communication may lead to a *should*-constraint on PM level, meaning that they should be in the same PM. Security concerns may necessitate a *must-not*-constraint on PM level, meaning that they must not be in the same PM etc.

A component may have a license assigned to it, if it is placement-relevant because the license fee is proportional to either the number of VMs or the number of PMs running components with the given license. For a VM-based license ℓ, let $V(\ell)$ denote the set of VMs containing at least one component associated with license ℓ, then the license fee to be paid because of ℓ is $fee(\ell) \cdot |V(\ell)|$. Similarly, if ℓ is a PM-based license and $P(\ell)$ denotes the set of PMs containing at least one component associated with ℓ, then the license fee to be paid because of ℓ is $fee(\ell) \cdot |P(\ell)|$. The total license fee to be paid is the sum of the fees for each license.

As a consequence, if multiple VMs containing components with the same license ℓ are in the same PM, then

- the license fee has to be paid only once if ℓ is a PM-based license;
- it has to be paid for each VM if ℓ is a VM-based license.

A PM may possess some PM features. A hardware (HW) affinity constraint can specify the relation of a component to a PM feature. The *type* of the HW affinity can be either *must* (the component definitely requires a PM with the given feature) or *should* (the component benefits from a PM with the given feature).

TABLE 1
Overview of used notation

Notation	Meaning
C	Set of all components
V	Set of all VMs
P	Set of all PMs
$v(c)$	VM hosting component c
$p(v)$	PM hosting VM v
d	Number of considered resource types
s_0	Size vector of an empty VM
$V(\ell)$	Set of VMs with at least one component associated with license ℓ
$P(\ell)$	Set of PMs with at least one component associated with license ℓ
$W(x)$	Power consumption of a PM with CPU load x
W_{min}	Minimum power consumption of a PM
W_{max}	Maximum power consumption of a PM

The power consumption of a PM is a function of its CPU load. As in several previous works [4], [18], [43], we use a linear approximation, i.e., the power consumption of a PM with CPU capacity c and CPU load x is given by

$$W(x) = W_{min} + (W_{max} - W_{min}) \cdot x/c,$$

where W_{min} and W_{max} are the minimum and maximum power consumption of the PM, respectively. Table 1 gives an overview of the used notation.

Now we summarize the problem's inputs, outputs, constraints, and objectives. The inputs are:

- the set of components with the associated colocation constraints, licenses, and HW affinities;
- the set of PMs with their PM features.

The output consists of

- the set of VMs to be used,
- the mapping of components to VMs,
- and the mapping of VMs to PMs.

The solution must respect the PM capacity constraints, the requirements of the components in terms of VM OS, the colocation constraints of type „must" and „must not," and the HW affinities of type „must." There are multiple objectives: minimizing the total energy consumption of the PMs, minimizing the total license fee, maximizing the number of satisfied colocation constraints of type „should" and „should not," and maximizing the number of satisfied HW affinities of type „should."

As with any model, this problem formulation also abstracts from some technical details. For example, the transient processes of turning a PM on or off, deploying a VM on a PM, or deploying a component in a VM are not considered, nor their overhead in terms of time and energy. This is in line with the problem formulations used in most previous works in this area [30] and represents a good approximation for long-running software components. For very dynamic settings, the problem formulation – and the subsequent algorithms – may need to be extended in this respect.

4 Mapping algorithms

In this section, we first devise an algorithm for the joint VM selection and VM placement problem, called Combined.

Then, for the purpose of comparison, we also introduce some algorithms for only VM selection respectively only VM placement.

VM placement and VM selection are tough combinatorial problems for which optimal methods are unfortunately intractable for practical problem sizes [31]. Therefore, in line with most existing works, the algorithms presented here are all heuristics.

4.1 Combined VM selection and VM placement

The aim of the COMBINED algorithm is to map a new component c on a VM and a PM. This can be done in several ways:

- Starting a new VM v to host c and placing v on a PM
- Selecting an existing VM v to host c and keeping v on the PM where it is
- Selecting an existing VM v to host c and migrating v to another PM

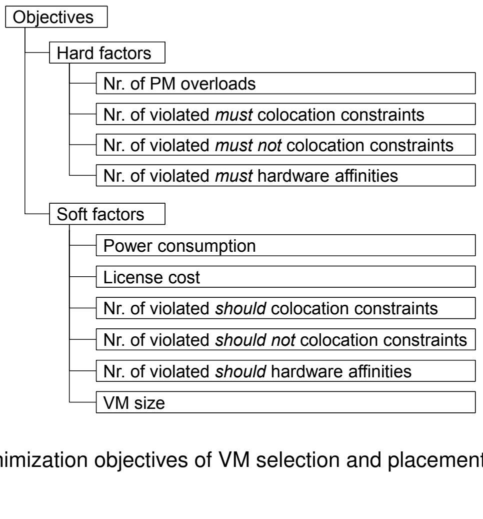
These three principal ways of mapping can be unified by considering all VMs from $V \cup \{v^*\}$ as possible hosts for c, where V is the set of existing VMs and v^* is a new VM, and considering all PMs as possible hosts for the selected VM. The basic idea of the COMBINED algorithm is to examine all these possibilities and choose the best one.

One challenge that needs to be tackled is the potentially large number of possible configurations to examine, namely $(|V|+1) \cdot |P|$. The naive approach of examining all possible configurations can be rather time-consuming if $|V|$ and $|P|$ are large, taking also into account that examining a possible configuration in terms of several objectives is also non-trivial. Note that this would still be a polynomial-time algorithm, but in order to quickly react to tenants' deployment requests, the selection and placement algorithm has to be fast even for large DCs.

For this reason, we decided to first filter the set of candidate PMs and VMs and take only the promising ones into account. As shown in Algorithm 1, we collect the promising VMs in a set V^* and the promising PMs in a set P^*. We start by placing a new VM with an appropriate OS in V^* (line 3). If there is a colocation constraint of type *must* or *should* between c and another component c', then either the VM or the PM hosting c' is also added, depending on whether it is a VM-level or PM-level colocation constraint (lines 4-12). Such VMs/PMs are indeed promising candidates, since mapping c onto them would satisfy the colocation constraint. Similarly, if c has a VM-based license, then all VMs containing a component with the same license are added to V^* (lines 13-18), whereas if c has a PM-based license, then all PMs containing a component with the same license are added to P^* (lines 19-26). These are again promising since mapping c onto them would incur no license fee. For hardware affinity constraints of c, all PMs offering the needed feature are added to P^* (lines 27-32). From all PMs where c would fit without overload, the ones with the highest load are also added to P^* (lines 33-35), since mapping c onto them would lead to good utilization and thus to relatively low energy consumption. In all these steps, if a PM p is added to P^*, then the VMs hosted on p are added to V^*. Finally, we add some further random VMs and PMs to V^* and P^* to extend the search space (lines 36-38). This is important especially if there are few VMs or few dependencies (colocation constraints, common licenses, affinity constraints). At the end, $V^* \times P^*$ defines the set of candidates to examine.

Algorithm 1 Determining candidate PMs and VMs

```
1:  procedure CANDIDATES($c$)
2:      $V^* \leftarrow \emptyset$, $P^* \leftarrow \emptyset$
3:      Add to $V^*$ a new VM with OS compatible with $c$
4:      for all must or should colocation constraint of $c$ do
5:          Let $c'$ be the other component of the constraint
6:          if VM-level colocation constraint then
7:              Add $v(c')$ to $V^*$
8:          else /* PM-level colocation constraint */
9:              Add $p(v(c'))$ to $P^*$
10:             Add each VM on $p(v(c'))$ to $V^*$
11:         end if
12:     end for
13:     if $c$ has a VM-based license $\ell$ then
14:         for all $v \in V$ do
15:             if there is a component in $v$ with license $\ell$ then
16:                 Add $v$ to $V^*$
17:             end if
18:         end for
19:     else if $c$ has a PM-based license $\ell$ then
20:         for all $p \in P$ do
21:             if there is a component in $p$ with license $\ell$ then
22:                 Add $p$ to $P^*$
23:                 Add each VM on $p$ to $V^*$
24:             end if
25:         end for
26:     end if
27:     for all hardware affinity constraint of $c$ do
28:         for all PM $p$ with the given PM feature do
29:             Add $p$ to $P^*$
30:             Add each VM on $p$ to $V^*$
31:         end for
32:     end for
33:     Let $P'$ be the set of PMs on which $c$ would fit
34:     Sort $P'$ in decreasing order of CPU load
35:     Add the first $k_1$ PMs of $P'$ to $P^*$ and their VMs to $V^*$
36:     Add $k_2$ random VMs to $V^*$
37:     Add $k_3$ random PMs that are on to $P^*$
38:     Add $k_4$ random PMs that are off to $P^*$
39:     return ($V^*$, $P^*$)
40: end procedure
```


Objectives
Hard factors
Nr. of PM overloads
Nr. of violated must colocation constraints
Nr. of violated must not colocation constraints
Nr. of violated must hardware affinities
Soft factors
Power consumption
License cost
Nr. of violated should colocation constraints
Nr. of violated should not colocation constraints
Nr. of violated should hardware affinities
VM size
Fig. 3. Minimization objectives of VM selection and placement

The other challenge in devising the COMBINED algorithm is rooted in the multi-objective nature of the problem: how to determine the best of the examined candidate configurations. As Fig. 3 shows, we differentiate between hard factors which should be 0 and soft factors that should be also minimized but with a lower priority than the hard factors.

The factors listed in Fig. 3, except for "VM size," directly relate to costs or constraint violations that need to be minimized. "VM size" has been included because, according to our preliminary experiments, otherwise the algorithm tends to colocate too many components in the same VM. This is logical since – because of the overhead of VMs – mapping the new component to an existing VM is always more energy-efficient than creating a new VM for it. However, having too large VMs may become a disadvantage in the long run, leading to fragmentation of the available PM capacity and hindering the colocation of future components with existing ones even if this were really necessary (because of colocation constraints or license fees). Therefore, since the algorithm makes online decisions based on current objective values without seeing the future, it was necessary to include VM size as a minimization objective to neutralize the energy bias and develop a more future-proof mapping.

For each examined candidate configuration and each optimization objective, we compute the difference that the given selection and/or placement decision would have on the given metric. Based on these atomic metrics, two compound metrics are computed for each examined candidate configuration: the sum of the hard factors and the weighted sum of the soft factors (cf. Fig. 3). For the soft factors, weighting is reasonable because power consumption values, license fees, VM sizes and numbers of violations are of different orders of magnitude, so they should be scaled to the same range to allow a meaningful comparison later on. The weight values should thus be chosen depending on the range of license costs, power consumption values etc. The weights can also be used to express differences in the importance of the individual soft factors. For the hard factors, weighting is not necessary (although possible) because all factors are numbers of violations.

To decide whether candidate configuration x is better than candidate configuration y, we use the following relation:

$$x \prec y \Leftrightarrow hard(x) < hard(y) \vee \\ \vee (hard(x) = hard(y) \wedge soft(x) < soft(y)),$$

where $hard(\cdot)$ and $soft(\cdot)$ denote the two compound metrics defined above.

Putting all pieces together, Algorithm 2 shows the body of the COMBINED algorithm. It should be noted how VM selection and VM placement are interleaved in this algorithm, since each examined configuration encodes both a VM selection and a VM placement decision.

4.2 Separate VM selection and VM placement

For comparison, we also develop two policies for VM selection (without VM placement) and two policies for VM placement (without VM selection). Any VM selection policy can be catenated with any VM placement policy, leading to four different algorithms for deploying a new component.

Algorithm 2 The COMBINED algorithm for adding a new component c

```
1: $(V^*, P^*) \leftarrow$ CANDIDATES($c$)
2: for all $v \in V^*$ do
3:     for all $p \in P^* \cup \{p(v)\}$ do
4:         Compute atomic objectives for $(v, p)$
5:         Compute compound objectives for $(v, p)$
6:     end for
7: end for
8: $(v, p) \leftarrow$ best examined configuration according to $\prec$
9: if $v$ is a new VM then
10:    Start new VM on $p$
11: else if $p(v) \neq p$ then
12:    Migrate $v$ from $p(v)$ to $p$
13: end if
14: Deploy $c$ on $v$
```

Algorithm 3 The INFORMED policy for selecting a VM for the new component c

```
1: Let $v^*$ be a new VM with OS compatible with $c$
2: $V^* \leftarrow V \cup \{v^*\}$
3: for all $v \in V^*$ do
4:     Compute atomic objectives for selecting $v$ for $c$
5:     Compute compound objectives for selecting $v$ for $c$
6: end for
7: $v \leftarrow$ best examined VM according to $\prec$
8: if $v = v^*$ then
9:     Start new VM
10: end if
11: Return $v$
```

4.2.1 DEDICATED *selection policy*

Our first VM selection policy always creates a new, dedicated VM for the new component. Despite its simplicity, this selection policy is quite powerful because it does not create any unnecessary dependence between components, thus leaving full flexibility to the subsequent placement as well as future re-optimizations by live migration. Accordingly, this approach has been used by some previous works [17], [21]. The obvious drawbacks of this policy are the relatively high overhead stemming from the high number of VMs and the lack of colocation for components that must or should be colocated.

4.2.2 INFORMED *selection policy*

To remedy the shortcomings of the DEDICATED selection policy, we devise a much more sophisticated policy aiming to make a well-informed decision on whether to colocate the new component with existing components or to deploy it in a new VM.

As shown in Algorithm 3, the INFORMED VM selection policy closely resembles the COMBINED algorithm. The differences stem directly from the fact that the INFORMED policy does not account for the placement: hence, it investigates only the possible VMs, not pairs of VMs and PMs. Note also that the INFORMED policy examines *all* the $|V| + 1$ possible VMs, whereas the COMBINED algorithm had to sample from its much larger search space to remain fast.

The biggest difference is in the way the objectives are computed. From the metrics shown in Fig. 3, the "Nr. of PM overloads," "Nr. of violated *must* / *should* hardware affinities," and "Power consumption" objectives are not applicable at the VM selection stage and are thus ignored in the INFORMED policy. As regards license costs, only VM-based licenses can be considered. VM-level colocation constraints can be fully evaluated, but concerning PM-level colocation constraints, we can only be sure about a violation in case of a *must not* or *should not* constraint (if the involved components are mapped to the same VM); for a PM-level *must* or *should* constraint, a violation cannot be determined at the VM selection stage. The "VM size" metric can be of course fully evaluated.

Because of the – soft – aim of minimizing VM size, components will be only colocated if this is necessary or advantageous for satisfying colocation constraints or for minimizing license fees.

4.2.3 BLACK-BOX *placement policy*

The placement policy receives as input the VM returned by the preceding VM selection policy, which may be a new or an existing VM. The placement policy determines a PM for this VM. In case of an existing VM, this means that the placement policy may decide to migrate the selected VM. This is in line with the COMBINED algorithm, which can also migrate the VM selected for the new component.

The BLACK-BOX placement policy does not consider the components within the VM to place, only its size. This is the same approach as taken by most previous works in the area of VM placement. As suggested by several researchers (e.g., Beloglazov and Buyya [5]), we use the best-fit heuristic to choose the PM that has enough capacity to host the VM but with the minimum remaining free capacity. The VM is then placed on this PM.

Recall that the capacity of a PM is a multi-dimensional vector. For comparing the free capacity of two PMs, we first convert them to single numbers. For this purpose, we take the minimum of the coordinates of the vector. In our previous work we also compared some other metrics for this purpose and found that the minimum metric gives good results [33].

Since this placement policy only considers the size of the VM, we can expect that it will lead to a good placement in terms of energy consumption and number of overloads, but will perform poorly in terms of license costs and conformance with colocation and hardware affinity constraints.

4.2.4 WHITE-BOX *placement policy*

To address the shortcomings of the BLACK-BOX placement policy, we devise a more sophisticated placement policy that also considers the relations of the components within the VM to be placed. Similarly to the INFORMED selection policy, the idea is again to mimic the COMBINED algorithm as much as possible, now at the level of VM placement.

As shown in Algorithm 4, this involves examining all PMs as possible hosts for the VM and choosing the best one in terms of the investigated objectives. From the objectives of Fig. 3, now all atomic metrics are relevant except for "VM size." In terms of license costs, only PM-based licenses are relevant at this stage; similarly, from the colocation constraints, only PM-level constraints are relevant. The other metrics are fully evaluated.

It should be noted that the INFORMED selection policy and the WHITE-BOX placement policy together base their decisions on the same set of information as the COMBINED algorithm and also the way they examine and compare possible candidates is analogous. However, there are two main differences. First, the COMBINED algorithm examines VM-PM pairs, i.e., it considers selection and placement together, whereas in the catenation of INFORMED and WHITE-BOX, first only VMs are considered until one VM is selected, and then only PMs are considered for the already selected VM. This can be seen as an advantage of the COMBINED algorithm. Second, both the INFORMED selection policy and the WHITE-BOX placement policy consider all their possible choices (all VMs respectively all PMs), whereas the COMBINED algorithm only examines a subset of the possible candidate configurations, so that it remains fast. The more thorough search can be seen as an advantage of the INFORMED and WHITE-BOX policies.

Algorithm 4 The WHITE-BOX policy for placing a VM v

```
 1: for all $p \in P$ do
 2:     Compute atomic objectives for placing $v$ on $p$
 3:     Compute compound objectives for placing $v$ on $p$
 4: end for
 5: $p \leftarrow$ best examined PM according to $\prec$
 6: if $v$ is a new VM then
 7:     Start new VM on $p$
 8: else if $p \neq p(v)$ then
 9:     Migrate $v$ from $p(v)$ to $p$
10: end if
```

5 EVALUATION

Our aim is to compare the different approaches to VM selection and VM placement:

- Decoupled VM selection and VM placement, as in most existing approaches (DEDICATED+BLACK-BOX)
- Partial integration:
  - VM selection also considers VM placement but not vice versa (INFORMED+BLACK-BOX)
  - VM placement also considers VM selection but not vice versa (DEDICATED+WHITE-BOX)
- Semi-integrated: VM selection considers VM placement and vice versa (INFORMED+WHITE-BOX)
- Fully integrated VM selection and VM placement (COMBINED)

5.1 Setup

Algorithms for VM placement and VM selection are usually evaluated either with a real cloud or by means of simulation. Using a real cloud is of course more realistic but it comes with several limitations. In particular, it is difficult to experiment with many different parameter settings or to scale the size of the experiment if a real cloud is used. Simulations are much more flexible and hence more popular for research on cloud resource management [38], [41], [48], [49]. Since we would like to compare several different algorithms under many different settings, a simulation-based approach is more appropriate. To still obtain practically relevant results, we used real-world test data, leading to a good compromise between a real cloud and pure simulation.

TABLE 2
Results for the base setup (component sizes only)

Algorithm	Energy [kWh]	Nr. of overloads	Nr. of migrations	Execution time [ms]
COMBINED	8,319.35	0	2,390.6	4.0
DEDICATED+BLACK-BOX	8,570.03	0	1,780.5	0.1
DEDICATED+WHITE-BOX	8,538.66	0	1,520.2	0.5
INFORMED+BLACK-BOX	8,567.11	0	1,792.3	0.3
INFORMED+WHITE-BOX	8,539.47	0	1,496.2	0.9

We have implemented all algorithms presented in Section 4 in a C++ program. To foster reproducibility, this program is freely available from https://sourceforge.net/p/vm-alloc/crosslayer.

In addition to the selection and placement algorithms discussed so far, the program also features a re-optimization algorithm which is invoked regularly and uses VM live migrations to adapt the placement to workload changes. The re-optimization algorithm works as follows: it takes a random VM and uses the WHITE-BOX placement policy to optimize its placement. This optimization step is repeated k_r times, where k_r is a given constant.

For component sizes, we used a real workload trace from the Grid Workloads Archive, namely the AuverGrid trace, available from http://gwa.ewi.tudelft.nl/datasets/gwa-t-4-auvergrid. From the trace, we used the first 10,000 tasks that had valid CPU and memory usage data. The simulated time (i.e., the time between the start of the first task and the end of the last one) is roughly one month, thus giving sufficient exposure to practical workload patterns.

As PMs, we simulated HP ProLiant DL380 G7 servers with Intel Xeon E5640 quad-core CPU and 16 GB RAM. Their power consumption varies from 280W (zero load) to 540W (full load) [23]. Throughout the experiments, we focus on two resource types: CPU and memory, i.e., $d = 2$. Concerning virtualization overhead, previous work reported 5-15% for the CPU [50] and 107-566 MB for memory [13]. In our experiments, we use 10% CPU overhead and 300 MB memory overhead. The VM placement is re-optimized every 5 minutes. Similarly, constraint violations are also checked every 5 minutes.

Each reported result is the average of 10 runs.

5.2 Component sizes only

In our first experiment, components are only characterized by their sizes, i.e., there are no license fees, colocation constraints, nor hardware affinities, and each component has the same OS. This is similar to the evaluation setup of most previous works.

The results – according to the relevant metrics – are shown in Table 2. As can be seen, all algorithms result in 0 overloads. In terms of energy consumption, the COMBINED algorithm has a clear advantage over the others; the results of the others are very close to each other. In particular, the used selection policy has practically no effect. This is indeed true because in this case the INFORMED policy has no reason to colocate multiple components in the same VM, hence it also starts a dedicated VM for each component. The WHITE-BOX placement policy performs slightly better than the BLACK-BOX policy. Since the components are characterized only by their sizes, there is not much difference between the two placement policies. The difference is only that BLACK-BOX uses the best-fit heuristic whereas WHITE-BOX chooses the PM based on its real power consumption.

The advantage of the COMBINED algorithm over the second best in terms of energy consumption is 219.31 kWh, or roughly 2.6%. The average electricity price in the Euro area for industrial customers amounted to 0.125 Euro per kWh in 2015.[3] Thus, the savings translate to 27.1 Euro. Scaling it to a data center with 10 thousand PMs, considering a 12-month period, and assuming a PUE (power usage effectiveness) of 1.7, which is a typical value[4], the total savings would amount to over 240,000 Euro per year.

In terms of the number of migrations (fourth column of Table 2), there is a clear difference between the algorithms. This could be important because too many migrations could lead to performance degradation or could even make the system unstable [16], [42]. However, relative to the length of the simulation, the number of migrations is actually quite low for all algorithms; even for the COMBINED algorithm which leads to the highest number of migrations, the average number of migrations per PM per day is only 3.32, which should not cause any problems.

Similarly, the COMBINED algorithm takes considerably more time (see last column of Table 2) than the other methods, but with an average execution time of 4.0 milliseconds, it can be still considered fast enough.

5.3 License fees

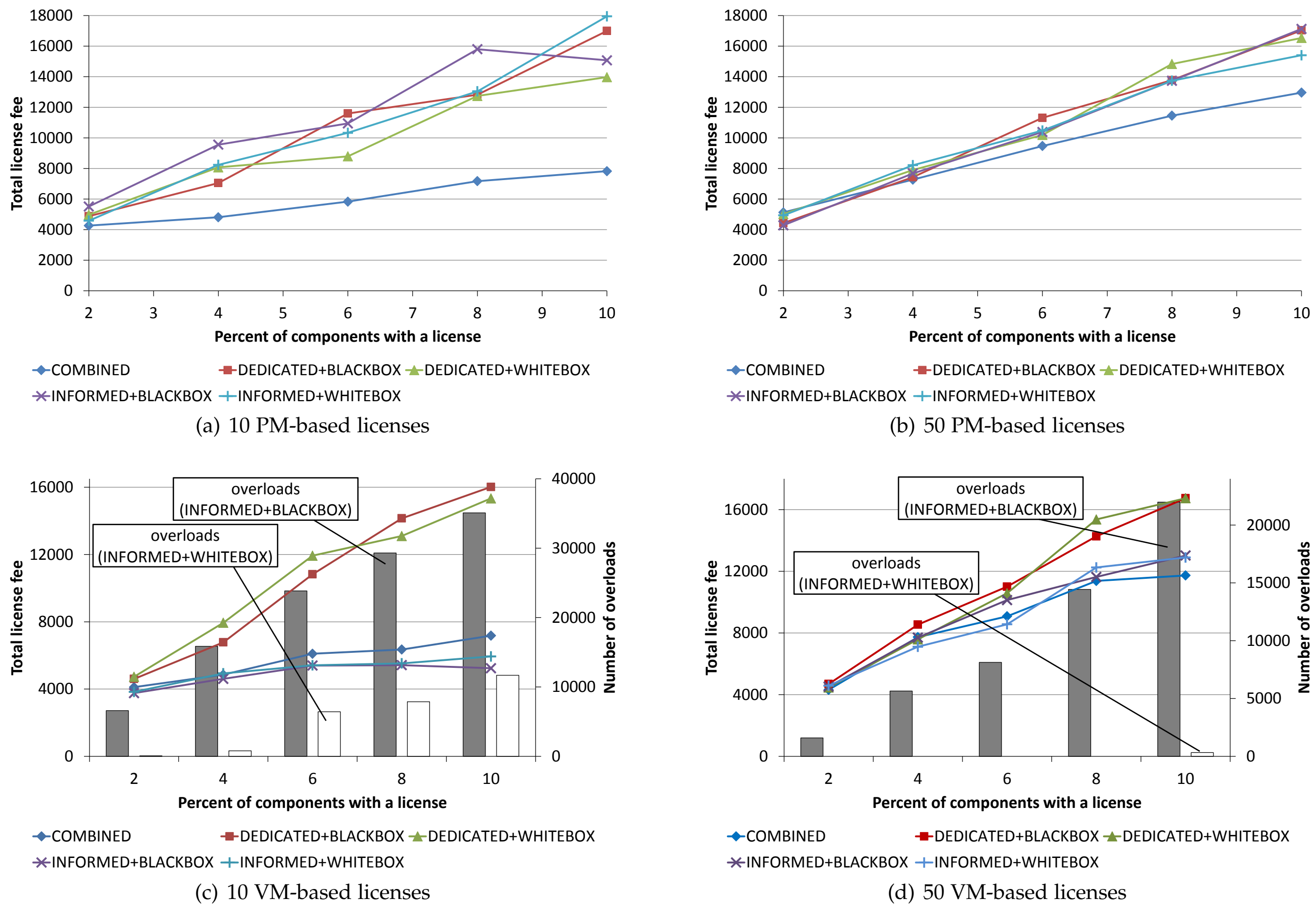
In the next set of experiments, we investigate the effect of license fees. For this purpose, the components are enriched with randomly generated license information.

First, we assume 10 different PM-based licenses (and no VM-based licenses). For each of them, the license fee is randomly chosen between 100 and 1000. We varied the number of components having a license from 2% to 10% of all components; each of the license-relevant components is associated to one of the 10 licenses, taken randomly. The resulting total license fees achieved by the different algorithms are depicted in Fig. 4(a). (Other metrics are not shown because they are very similar to the values from Table 2; in particular, all algorithms lead to 0 overloads.)

The figure clearly shows the superiority of the COMBINED algorithm over all others. The difference keeps growing with increasing number of license-relevant components; if 10% of all components have a PM-based license, then the COMBINED algorithm achieves 44% lower license fees than the best result of the other algorithms. Among the other algorithms, there is no clear winner.

Fig. 4(b) shows the results of the same experiment but with 50 instead of 10 PM-based licenses. Again, the COMBINED algorithm leads to the best results in most cases and its advantage grows with increasing number of license-

3. http://ec.europa.eu/eurostat/statistics-explained/index.php/Energy_price_statistics
4. http://www.datacenterknowledge.com/archives/2014/06/02/survey-industry-average-data-center-pue-stays-nearly-flat-four-years/



(a) 10 PM-based licenses

(b) 50 PM-based licenses

(c) 10 VM-based licenses

(d) 50 VM-based licenses

Fig. 4. License costs achieved by the different algorithms for an increasing number of components with licenses

relevant components[5]. However, its advantage over the other algorithms is significantly smaller in this case. This is because now the number of different licenses is greater, hence – given the same number of license-relevant components – the number of components with the same license is smaller, leading to less opportunities for cost saving by colocation of components with the same license.

Next, we used VM-based instead of PM-based licenses. The results for 10 VM-based licenses are shown in Fig. 4(c). The figure shows the license fees for all algorithms; moreover, it shows the number of overloads (as bars) for those algorithms where this was not 0. As can be seen, DEDICATED leads to high license fees because it does not colocate components, hence the license fee has to be paid for every component. COMBINED achieves much lower license fees (again, with a growing advantage). INFORMED leads to even lower license fees. However, this is the result of too aggressive colocation of components with the same license fee, leading to VMs whose sizes surpass the PMs' capacity, resulting in overloads. COMBINED finds a good balance between avoiding overloads and minimizing license fees.

5. For the extreme case when only 2% of components have a license, the COMBINED algorithm is not the winner anymore. In this case, most decisions of the COMBINED algorithm are governed by other objectives, e.g., optimizing energy consumption. To that end, it puts multiple components into VMs to achieve better utilization. This, however, can lead to big VMs that cannot be colocated with others even if this would be beneficial from a license fee point of view.

Increasing the number of (VM-based) licenses from 10 to 50 leads to a similar pattern, with smaller differences, as shown in Fig. 4(d). DEDICATED again performs worst, but the three other algorithms lead to similar results in terms of license fees. Also, as long as the number of license-relevant components is not too high, INFORMED+WHITE-BOX does not generate overloads, hence it could be seen as a good alternative to COMBINED. However, for 10% license-relevant components, the number of overloads is already non-zero.

It is interesting to compare the relative performance of COMBINED and INFORMED between Figures 4(c) and 4(d). In Fig. 4(c), the number of components with the same license is high, leading to significant optimization opportunities that INFORMED fully leverages, resulting in lower license fees – although at the cost of PM overloads – than achieved by COMBINED. This is a clear trend for 6%, 8%, and 10% of components having a license. In Fig. 4(d), the number of components with the same license is much smaller, leading to less optimization opportunities. Thus, the results of the different algorithms are closer to each other. The results of INFORMED and COMBINED are not significantly different: sometimes one of them is a bit lower, other times the other is a bit lower, without a clear winner.

5.4 Colocation constraints

The next set of experiments evaluates the impact of colocation constraints. In each experiment, 100 colocation constraints were generated for randomly selected components.

TABLE 3
Number of constraint violations plus number of overloads for different colocation constraints

Algorithm	PM-level				VM-level				All
	must	*should*	*should not*	*must not*	*must*	*should*	*should not*	*must not*	
COMBINED	1,956.9	140.1	0	0	1,728.6	137.1	0	0	449.3
DEDICATED+BLACK-BOX	12,541.8	12,321.9	1,572.9	1,573.5	17,726.5	17,491.9	0	0	8,969.7
DEDICATED+WHITE-BOX	13,752.2	10,585.6	0	0	16,899.5	17,105.7	0	0	7,587.0
INFORMED+BLACK-BOX	14,878.7	13,587.3	1,615.5	1,399.1	8,391.3	13,473.9	0	0	6,116.5
INFORMED+WHITE-BOX	13,416.7	11,141.3	0	0	953.2	1,179.3	0	0	6,258.3

Table 3 shows the results in a condensed form. Each column corresponds to one experiment. For example, in the experiment of the second column, all colocation constraints were PM-level and of type *must*; in the 8th column, all colocation constraints were VM-level and of type *should not*. While in most columns, all colocation constraints were on the same level and of the same type, the last column is different: it is a mix of the 8 combinations of colocation level and type, where each combination is present with approximately the same number of constraints. For each experiment, we report the sum of the number of colocation constraint violations and the number of overloads.

For PM-level *must* and *should* colocation constraints (second and third column of the table), the COMBINED algorithm is clearly superior to all others, and all other algorithms achieve similarly poor results. Looking more precisely into the operation of the algorithms, the following can be understood about the reasons:

- The INFORMED selection policy has no incentive to colocate multiple components in the same VM since in these experiments the colocation constraints are all on the PM level. As a result, it creates a dedicated VM for each component. This is why there is no significant difference between the results of the two VM selection policies.
- Since both VM selection policies create small VMs, this leads to low fragmentation. Therefore, when a PM-level colocation constraint motivates the WHITE-BOX placement algorithm to place the new VM on the PM where one of the already placed components resides, it will often not succeed because the given PM does not have sufficient free capacity. This is why there is no significant difference between the results of the two VM placement policies.
- The COMBINED algorithm on the other hand, when confronted with this situation, will put the new component into the same VM as its peer and then migrate the VM containing both components to a PM with sufficient free capacity. Note that the other approaches also have this option, but do not choose it because of the separate evaluation of the selection and placement possibilities.

Concerning the PM-level *should not* and *must not* constraints, the results are more easily understood. The COMBINED algorithm as well as the WHITE-BOX placement policy are able to avoid constraint violations by not placing the new component (respectively the VM where it has been put) onto the same PM as some other component(s). The BLACK-BOX placement policy, which does not consider colocation constraints, necessarily leads to some violations. It is interesting to note that the number of violations is now much lower than in the case of *must* and *should* constraints. This is not surprising though: placing the new VM on a random PM has a high chance to meet a *should not* or *must not* constraint if the number of "bad" PMs is low, whereas meeting a *must* or *should* constraint has much lower probability.

For VM-level *must* and *should* colocation constraints, the DEDICATED VM selection policy obviously leads to poor results since it never selects the same VM for components that should be colocated. In fact, these results are even significantly worse than in the case of the similar PM-level constraints, since in this case definitely all constraints will be violated, whereas in the case of PM-level constraints, PM-level colocation was still possible. The INFORMED selection policy, on the other hand, puts all components that must or should be in the same VM indeed into the same VM. Together with the WHITE-BOX placement policy, this leads to very good results, similar to those of the COMBINED algorithm. For VM-level *must* colocation constraints, it even improves on the results of the COMBINED algorithm.

For VM-level *should not* and *must not* colocation constraints, as can be seen, all tested algorithms achieve optimal results. Not colocating some components in the same VM is very easy, for example by using dedicated VMs for each component.

Over all experiments with colocation constraints, the COMBINED algorithm gives excellent results: with the exception of the VM-level *must* constraints, where it ranks only second after the INFORMED+WHITE-BOX combination, it always gives the best results, in several cases dramatically better results than any other algorithm. This is also mirrored in the last column of Table 3, showing the combined effect of different colocation constraints. Here, too, the COMBINED algorithm is the clear winner, leading to more than an order of magnitude better results than all other algorithms. From the results of the remaining algorithms it is also apparent that the INFORMED VM selection policy has a clear advantage over the DEDICATED policy. This is not surprising, given the inability of the DEDICATED policy to appropriately handle VM-level *must* or *should* colocation constraints.

5.5 Hardware affinity

In this set of experiments, the PMs were enriched with PM feature information and the components were enriched with hardware affinity requirements. In particular, we define k_f PM features and each PM has each feature with probability p_f. Each component requires (*must* relationship) each PM

TABLE 4
Number of hard hardware affinity constraint violations plus number of overloads, depending on the number of PM features

Algorithm	Nr. of PM features					
	1	2	3	4	5	6
COMBINED	0	322	0	97	140	425
DEDICATED+BLACK-BOX	11,745	24,589	32,780	42,621	56,264	71,904
DEDICATED+WHITE-BOX	0	0	0	343	0	0
INFORMED+BLACK-BOX	11,973	22,961	36,385	46,647	60,120	69,419
INFORMED+WHITE-BOX	0	0	0	643	121	8

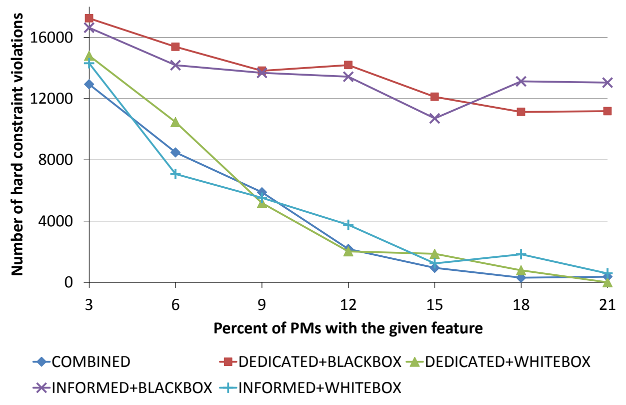


Fig. 5. Number of hard hardware affinity constraint violations plus number of overloads, depending on the percent of PMs with the required feature

feature with probability p_r respectively benefits from it (*should* relationship) with probability p_b.

First, we fix $p_f = 20\%$, $p_r = 1\%$, $p_b = 5\%$, and vary k_f from 1 to 6. The results are shown in Table 4. As can be seen, the BLACK-BOX placement policy leads to poor results with both selection policies. This is logical since the selection policies cannot do anything to fulfill hardware affinity constraints, and the BLACK-BOX placement policy does not look at the hardware affinity requirements of the components within the VM to be placed. On the other hand, the WHITE-BOX placement policy, which does account for the hardware affinity requirements of the components within the VM to be placed, achieves excellent results, irrespective of the used VM selection policy. The COMBINED algorithm achieves similarly good results. It should also be noted that increasing k_f leads to a higher number of hardware affinity requirements; for the BLACK-BOX placement policy, this also translates into more constraint violations. However, for the other algorithms, the number of constraint violations remains low.

In the next experiment, we considered a single PM feature and varied the percentage of PMs offering this feature from 3% to 21%. In each run, 1% of all components required the given feature while 5% of all components were defined to benefit from it. The results, in terms of the number of hard constraint violations (i.e., number of violations of hard hardware affinity constraints plus number of overloads) are shown in Fig. 5.

As can be seen, if only few PMs possess the required feature, then all algorithms lead to a high number of violations; it is probably not even theoretically possible to fulfill all hardware affinity requirements. As the number of PMs with

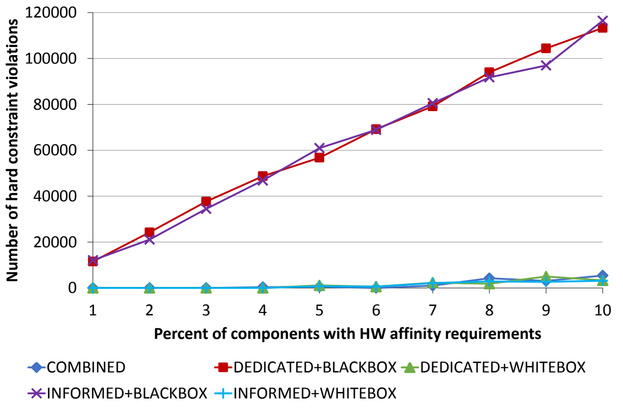


Fig. 6. Number of hard hardware affinity constraint violations plus number of overloads, depending on the fraction of components with (hard) hardware affinity requirements

the given feature increases, the results form two clusters: the COMBINED algorithm and the algorithms using the WHITE-BOX placement policy manage to use these PMs to fulfill a growing number of hardware affinity requirements, leading to a steady decrease in the number of hard constraint violations. On the other hand, the BLACK-BOX policy hardly benefits from the increased availability of "good" PMs, in line with the fact that this VM placement policy does not explicitly consider hardware affinities. Within the two clusters, there are no significant differences, suggesting that VM selection does not considerably influence the satisfaction of hardware affinity requirements.

A very similar pattern can be observed also in Fig. 6. In this experiment, a single PM feature is considered which is offered by 20% of all PMs. The fraction of components requiring the given feature is varied from 1% to 10%. As can be seen in the figure, the BLACK-BOX VM placement policy again leads to a high number of constraint violations, which steeply increases with the growing number of components with hardware affinity requirements. On the other hand, the WHITE-BOX placement algorithm and the COMBINED algorithm can solve the problem with a much lower number of constraint violations.

5.6 Number of operating systems

We also varied the number of operating systems and let each component require a randomly selected OS. However, the number of the considered operating systems did not have a noticeable effect on the used quality metrics.

5.7 Putting the pieces together

So far, the effect of different aspects was investigated in isolation. In practice, all aspects may be present at the same time. Therefore, we also present the results of an experiment in which multiple aspects are applied as follows:

- License fees:
  - Number of PM-based licenses: 10
  - Number of VM-based licenses: 10
  - Ratio of components with a license: 5%
- Colocation constraints:

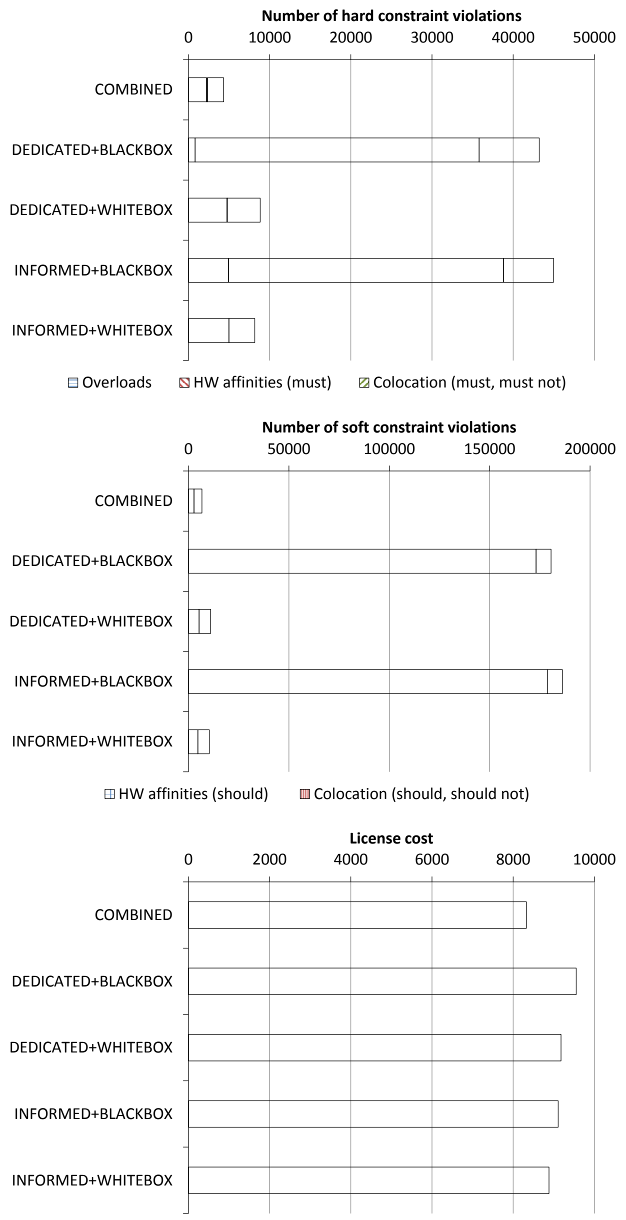


Fig. 7. Results with license fees, colocation constraints, and hardware affinities together

- Number of PM-level constraints: 100
- Number of VM-level constraints: 100
- Ratio of constraints with type *must*, *should*, *should not*, *must not*: 25% each

- Hardware affinities:
  - Number of PM features: 3
  - For each PM feature, ratio of PMs having the feature: 20%
  - For each PM feature, ratio of components requiring the feature: 1%
  - For each PM feature, ratio of components benefiting from the feature: 5%
- Number of operating systems: 3

The results are shown in Fig. 7 according to three dimensions: hard constraint violations, soft constraint violations, and license costs. As can be seen, the BLACK-BOX VM placement policy is not competitive in terms of both hard and soft constraint violations, no matter which VM selection policy is used. From the remaining three algorithms, the COMBINED algorithm delivers the best result according to all three dimensions. The INFORMED+WHITE-BOX algorithm is second according to all dimensions, with 89% more hard constraint violations, 55% more soft constraint violations, and 7% higher license costs than the COMBINED algorithm. Finally, the DEDICATED+WHITE-BOX algorithm leads to 8% more hard constraint violations, 6% more soft constraint violations, and 3% higher license costs than the INFORMED+WHITE-BOX algorithm.

TABLE 5
Effect of search space limitation: results relative to COMBINED

Metric	Relative result of COMBINED-FULL
Hard violations	-5.5%
Soft violations	-0.6%
License costs	+5.6%
Execution time	+671%

5.8 Effect of search space limitation

As explained in Section 4.1, the COMBINED algorithm considers only a subset of all possible PM-VM pairs, so that it remains fast. Now we investigate the effect of this limitation by comparing it against a version of the algorithm, called COMBINED-FULL, that considers all possible PM-VM pairs[6].

We have repeated the experiment of Section 5.7 also with the COMBINED-FULL algorithm. As Table 5 shows, COMBINED-FULL achieves an improvement of 5.5% in the number of violations of hard constraints over COMBINED. However, this comes at the expense of a comparable increase in license costs as well as a more than 7 times increase in the algorithm's execution time. This shows that the method used in COMBINED to determine sensible candidate solutions represents a good trade-off between execution time and solution quality.

6 CONCLUSIONS

Based on the measurement results of Section 5, we can now try to answer the original questions regarding the benefits of (i) doing VM selection and VM placement together and (ii) including information about VM placement in VM selection and about VM selection in VM placement. For this purpose, we first summarize the empirical results:

- For PM-based licenses, the COMBINED algorithm results in up to 44% lower license fees than the second best algorithm.
- For VM-based licenses, the COMBINED algorithm and the INFORMED VM selection policy lead to significantly lower license fees than the DEDICATED policy. However, in the case of the INFORMED policy, this comes at the cost of PM overloads.

6. It should be noted that also COMBINED-FULL is a heuristic. It considers all possible solutions that use a single migration, but it does not consider performing multiple migrations.

- For PM-level *must* and *should* colocation constraints, COMBINED leads to respectively 84% and 99% improvement over the second best algorithm.
- For PM-level *should not* and *must not* colocation constraints, the COMBINED algorithm and the WHITE-BOX VM placement policy lead to clearly better results than the BLACK-BOX placement policy.
- For VM-level *must* and *should* colocation constraints, the COMBINED algorithm and the INFORMED+WHITE-BOX algorithm lead to considerably better results than the other algorithms.
- In the presence of hardware affinity constraints, the COMBINED algorithm and the WHITE-BOX VM placement policy result in up to 97% better results than the BLACK-BOX placement policy.
- When all types of constraints were applied at once, the BLACK-BOX policy performed poorly; from the other algorithms, COMBINED was clearly the best, INFORMED+WHITE-BOX was second, and DEDICATED+WHITE-BOX was the third according to all considered dimensions.
- The COMBINED algorithm leads to 2-3% lower energy consumption than the four other tested methods. (Although not shown explicitly, this holds consistently for all cases without overloads.)

Altogether, we can conclude that the COMBINED algorithm is in all cases among the best-performing algorithms. Sometimes it clearly outperforms all other methods (e.g., for PM-based licenses), in other cases it is just one of the best performers. However, in none of the test cases was it clearly inferior to another algorithm. So the answer to the first question is clear: combining VM selection and VM placement in a single optimization algorithm leads to significant benefits compared to the isolated treatment of the two problems as in most existing works (DEDICATED+BLACK-BOX).

Regarding the second question, we can state that the WHITE-BOX VM placement policy clearly outperforms the BLACK-BOX policy. Hence, it is advantageous to include component-level information in VM placement decisions. The relationship between the DEDICATED and INFORMED VM selection policies is less clear because in many cases, the difference between their results was marginal. However, there were some test cases where INFORMED led to clearly better results: for VM-level *must* and *should* colocation constraints and, even more importantly, when all types of constraints were applied at once. Thus we conclude that, for these types of scenarios, it is also advantageous to include in VM selection decisions foresight into VM placement.

In terms of the current state of the art, it should be underlined that existing approaches almost exclusively focus either on VM selection or VM placement, ignoring the other problem. Hence, the DEDICATED+BLACK-BOX algorithm embodies the current state of the art. Compared to this, our results show that the tighter the two problems are integrated, the better results can be achieved.

The comparison between the fully integrated approach (COMBINED) and the semi-integrated approach (INFORMED+WHITE-BOX) is tricky because none of them dominates the other in all considered metrics. COMBINED was better in handling licenses and PM-level colocation constraints, the two approaches yielded similar results for handling VM-level colocation constraints and HW affinity constraints, while INFORMED+WHITE-BOX led to fewer migrations and had lower execution time. The number of migrations and the execution time of COMBINED are also in an acceptable range, and so, since it leads to lower license costs and fewer constraint violations, we consider it preferable. Also in the experiment which contained all the investigated aspects, the COMBINED approach led to better results according to all measured metrics.

A further question is whether there is any "hidden" overhead of the proposed approach, since it leads to more co-locations and as VM isolation is not perfect, this could lead to performance degradation. Fortunately, our model supports anti-colocation constraints with which co-location of components that would interfere with each other can be prohibited. Therefore, the proposed approach will only colocate components that do not interfere.

ACKNOWLEDGMENTS

This work was partially supported by the Hungarian Scientific Research Fund (Grant Nr. OTKA 108947) and the European Union's Horizon 2020 research and innovation programme under grant 731678 (RestAssured).

REFERENCES

[1] P. Arroba, J. M. Moya, J. L. Ayala, and R. Buyya, "Dynamic voltage and frequency scaling-aware dynamic consolidation of virtual machines for energy efficient cloud data centers," *Concurrency and Computation: Practice and Experience*, 2016.
[2] J. Bao, Z. Lu, J. Wu, S. Zhang, and Y. Zhong, "Implementing a novel load-aware auto scale scheme for private cloud resource management platform," in *Network Operations and Management Symposium (NOMS)*, 2014.
[3] D. Bartók and Z. A. Mann, "A branch-and-bound approach to virtual machine placement," in *Proceedings of the 3rd HPI Cloud Symposium "Operating the Cloud"*, 2015, pp. 49–63.
[4] 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, pp. 755–768, 2012.
[5] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.
[6] ——, "Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1366–1379, 2013.
[7] J. Betz, D. Westhoff, and G. Müller, "Survey on covert channels in virtual machines and cloud computing," *Transactions on Emerging Telecommunications Technologies*, 2017.
[8] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera, "A stable network-aware VM placement for cloud systems," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2012, pp. 498–506.
[9] L. F. Bittencourt, E. R. Madeira, and N. L. da Fonseca, "Scheduling in hybrid clouds," *IEEE Communications Magazine*, vol. 50, no. 9, pp. 42–47, 2012.
[10] L. F. Bittencourt, R. Sakellariou, and E. R. M. Madeira, "Using relative costs in workflow scheduling to cope with input data uncertainty," in *Proceedings of the 10th International Workshop on Middleware for Grids, Clouds and e-Science*, 2012.
[11] D. Breitgand and A. Epstein, "SLA-aware placement of multi-virtual machine elastic services in compute clouds," in *12th IFIP/IEEE International Symposium on Integrated Network Management*, 2011, pp. 161–168.

[12] D. Candeia, R. Araújo, R. Lopes, and F. Brasileiro, "Investigating business-driven cloudburst schedulers for e-science bag-of-tasks applications," in *2nd IEEE International Conference on Cloud Computing Technology and Science*, 2010, pp. 343–350.

[13] C. R. Chang, J. J. Wu, and P. Liu, "An empirical study on memory sharing of virtual machines for server consolidation," in *IEEE 9th International Symposium on Parallel and Distributed Processing with Applications*, 2011, pp. 244–249.

[14] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See spot run: using spot instances for mapreduce workflows," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (HotCloud'10)*, 2011.

[15] M. R. Chowdhury, M. R. Mahmud, and R. M. Rahman, "Study and performance analysis of various VM placement strategies," in *16th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE, 2015.

[16] U. Deshpande and K. Keahey, "Traffic-sensitive live migration of virtual machines," in *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2015, pp. 51–60.

[17] R. Ganesan, S. Sarkar, and A. Narayan, "Analysis of SaaS business platform workloads for sizing and collocation," in *IEEE 5th International Conference on Cloud Computing (CLOUD)*, 2012, pp. 868–875.

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

[19] T. A. L. Genez, L. F. Bittencourt, and E. R. M. Madeira, "Workflow scheduling for SaaS/PaaS cloud providers considering two SLA levels," in *Network Operations and Management Symposium (NOMS)*. IEEE, 2012, pp. 906–912.

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

[21] P. Hoenisch, D. Schuller, S. Schulte, C. Hochreiner, and S. Dustdar, "Optimization of complex elastic processes," *IEEE Transactions on Services Computing*, vol. 9, no. 5, pp. 700–713, 2016.

[22] P. Hoenisch, I. Weber, S. Schulte, L. Zhu, and A. Fekete, "Four-fold auto-scaling on a contemporary deployment platform using docker containers," in *International Conference on Service-Oriented Computing*, 2015, pp. 316–323.

[23] HP, "Power efficiency and power management in HP ProLiant servers," http://h10032.www1.hp.com/ctg/Manual/c03161908.pdf, 2012.

[24] D. Lago, E. Madeira, and L. Bittencourt, "Power-aware virtual machine scheduling on clouds using active cooling control and DVFS," in *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*, 2011.

[25] U. Lampe, T. Mayer, J. Hiemer, D. Schuller, and R. Steinmetz, "Enabling cost-efficient software service distribution in infrastructure clouds at run time," in *IEEE International Conference on Service-Oriented Computing and Applications*, 2011.

[26] U. Lampe, M. Siebenhaar, R. Hans, D. Schuller, and R. Steinmetz, "Let the clouds compute: cost-efficient workload distribution in infrastructure clouds," in *Proceedings of the 9th International Conference on Economics of Grids, Clouds, Systems, and Services (GECON 2012)*. Springer, 2012, pp. 91–101.

[27] P. Leitner and J. Cito, "Patterns in the chaos – a study of performance variation and predictability in public IaaS clouds," *ACM Transactions on Internet Technology*, vol. 16, no. 3, 2016.

[28] W. Li, J. Tordsson, and E. Elmroth, "Virtual machine placement for predictable and time-constrained peak loads," in *Proceedings of the 8th International Conference on Economics of Grids, Clouds, Systems, and Services (GECON 2011)*. Springer, 2011, pp. 120–134.

[29] ——, "Modeling for dynamic cloud scheduling via migration of virtual machines," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science*, 2011, pp. 163–171.

[30] Z. A. Mann, "Allocation of virtual machines in cloud data centers – a survey of problem models and optimization algorithms," *ACM Computing Surveys*, vol. 48, no. 1, 2015.

[31] ——, "Approximability of virtual machine allocation: much harder than bin packing," in *Proceedings of the 9th Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, 2015, pp. 21–30.

[32] ——, "A taxonomy for the virtual machine allocation problem," *International Journal of Mathematical Models and Methods in Applied Sciences*, vol. 9, pp. 269–276, 2015.

[33] ——, "Interplay of virtual machine selection and virtual machine placement," in *Proceedings of the 5th European Conference on Service-Oriented and Cloud Computing*, 2016, pp. 137–151.

[34] M. Mishra and A. Sahoo, "On theory of VM placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach," in *IEEE International Conference on Cloud Computing*, 2011, pp. 275–282.

[35] D. Oliveira, K. A. C. S. Ocana, F. Baiao, and M. Mattoso, "A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds," *Journal of Grid Computing*, vol. 10, pp. 521–552, 2012.

[36] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "Efficient virtual machine sizing for hosting containers as a service," in *IEEE World Congress on Services*, 2015, pp. 31–38.

[37] A. Radhakrishnan and V. Kavitha, "Energy conservation in cloud data centers by minimizing virtual machines migration through artificial neural network," *Computing*, vol. 98, no. 11, pp. 1185–1202, 2016.

[38] S. Rampersaud and D. Grosu, "Sharing-aware online virtual machine packing in heterogeneous resource clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 2046–2059, 2017.

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

[40] M. Sedaghat, F. Hernandez-Rodriguez, and E. Elmroth, "A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, 2013, article nr. 6.

[41] L. Shi, Z. Zhang, and T. Robertazzi, "Energy-aware scheduling of embarrassingly parallel jobs and resource allocation in cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 6, pp. 1607–1620, 2017.

[42] W. Song, Z. Xiao, Q. Chen, and H. Luo, "Adaptive resource provisioning for the cloud using online bin packing," *IEEE Transactions on Computers*, vol. 63, no. 11, pp. 2647–2660, 2014.

[43] P. Svärd, W. Li, E. Wadbro, J. Tordsson, and E. Elmroth, "Continuous datacenter consolidation," in *IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, 2015, pp. 387–396.

[44] L. Tomás and J. Tordsson, "An autonomic approach to risk-aware data center overbooking," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 292–305, 2014.

[45] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 358–367, 2012.

[46] D. Villegas, A. Antoniou, S. M. Sadjadi, and A. Iosup, "An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2012, pp. 612–619.

[47] S. Wang, A. Zhou, C.-H. Hsu, X. Xiao, and F. Yang, "Provision of data-intensive services through energy- and QoS-aware virtual machine placement in national cloud data centers," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 2, pp. 290–300, 2016.

[48] Z. Wang, M. M. Hayat, N. Ghani, and K. B. Shaban, "Optimizing cloud-service performance: Efficient resource provisioning via optimal workload allocation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 6, pp. 1689–1702, 2017.

[49] Y. Xia, M. Tsugawa, J. A. B. Fortes, and S. Chen, "Large-scale VM placement with disk anti-colocation constraints using hierarchical decomposition and mixed integer programming," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1361–1374, 2017.

[50] Y. Zhou, Y. Zhang, H. Liu, N. Xiong, and A. V. Vasilakos, "A bare-metal and asymmetric partitioning approach to client virtualization," *IEEE Transactions on Services Computing*, vol. 7, no. 1, pp. 40–53, 2014.

Zoltán Ádám Mann received the MSc and PhD degrees in Computer Science from Budapest University of Technology and Economics in 2001 and 2005, respectively. He is currently senior researcher at paluno – The Ruhr Institute for Software Technology, University of Duisburg-Essen. His research interests include optimization problems and algorithms in cloud computing.

Resource optimization across the cloud stack —Supplemental material—

Zoltán Ádám Mann

A Aspects of the component–VM–PM mapping

Depending on the characteristics of the application components, the used VM selection and placement strategies, and general DC management policies, there can be many different aspects that must be taken into account in the trilateral component–VM–PM mapping. In the following, we provide an analysis of the most important aspects, grouped into three categories: aspects relating to VM sizing, co-location, and other affinity aspects.

A.1 VM sizing

The following aspects of the trilateral component–VM–PM mapping must be taken into account in relation with the sizing of VMs:

- **Placeability.** As demonstrated by the example of the Introduction in the paper, the way components are mapped to VMs influences the size of the resulting VMs, which in turn determines the possible placements of the VMs on the PMs and hence the costs of the placement. It is difficult to make good decisions in VM selection because the placeability of a VM depends not only on its own size, but also on the size of other – existing and future – VMs.
- **Possibilities of live migration.** VMs provide a generic and transparent facility for migration in order to adapt to changes in the workload. Although some applications may offer the possibility to move individual components from one VM to another, this cannot be assumed in general. Hence in this paper we assume that only VMs can be migrated. The VMs are thus the unit of migration, which means that the VMs' sizes determine the granularity of migrations [4]. From this point of view, it is beneficial to have small VMs because they allow a fine-granular control of the PMs' utilization, thus avoiding fragmentation and achieving near-optimal utilization.
- **Overhead.** Virtualization introduces some overhead in terms of resource consumption. Since every VM adds some overhead (e.g., the size of the guest operating system), from this point of view it is beneficial to have a lower number of larger VMs. Obviously, this contradicts the above aspect which would lead to many small VMs, so that a good balance has to be found between the two aspects.

A.2 Co-location

The following aspects all lead to some kind of constraints or preferences on co-location of components in the same VM and/or PM:

- **Communication.** For components that communicate with each other, it may be necessary or at least advantageous to map them to the same VM, or at least to VMs on the same PM [5]. For example, a legacy application (i.e., one that was not developed with a multi-VM deployment in mind) may consist of multiple components that communicate via shared memory; in this case, these components have to be mapped to the same VM. If they communicate for instance through TCP sockets, then such a colocation is not necessary, but for communication-intensive applications it may still be advantageous in order to reduce latency and save network bandwidth.
- **Security.** Although virtualization provides a level of isolation between co-located VMs, it does not provide sufficient defense against malicious attacks. Using covert channels in hardware or software, a malicious VM can gain sensitive information from co-located VMs [2]. Therefore it is important to co-locate VMs hosting critical components only with VMs hosting only trusted components. In a private cloud, security concerns are typically lower than in public clouds, but in critical domains (e.g., banking), it is still important to isolate critical productive components from the ones whose security is not guaranteed.
- **Fault tolerance.** For some components, high availability may be necessary to improve reliability [6]. This has two important consequences for selection and placement. First, such components should not share a VM with other, less stable components to avoid that the failure of another component crashes the VM. (In contrast, they may be on the same PM since a VM can tolerate the crashing of a co-located

VM.) Second, if a component is replicated to guarantee high availability, then the replicated instances should be placed on different PMs so that a PM fault impacts only a single instance.

- **Performance interference.** Components in the same VM compete directly for all system resources. Between VMs, the virtualization layer partitions some resources (e.g., memory space), but this isolation is far from perfect, so that for some resources – like memory bandwidth or caches – there is also competition between co-located VMs, which may lead to significant performance degradation [1]. Therefore, components that use the same resource intensively, should be packed into distinct VMs, possibly also on distinct PMs.
- **Correlated load peaks.** Sudden load increases are dangerous in servers with high utilization because resource overloads may easily lead to SLO violations. It is especially problematic if the load of multiple components on the same PM or VM increases at the same time. Therefore, co-location of several correlated components should be avoided [3].

A.3 Other affinity aspects

The following aspects, leading to software or hardware affinity preferences, must also be taken into account when mapping components to VMs and VMs to PMs:

- **Operating system dependency.** In many cases, application components depend on a specific operating system (OS), a specific version (or range of versions) of an OS, or are compatible only with a set of OSs. While the OS of the VM can be chosen independently from the host OS, the OS of the VM must match the OS requirements of the components.
- **Hardware affinity.** Some components may require some special hardware feature, or there may be a preference for such. For example, a component may run only on a PM with a GPU[1] of a given vendor, or it may be able to take advantage of such hardware to boost its performance.
- **Licensing.** Some components may require costly licenses. There are many licensing models, several of which are agnostic of placement: e.g., if license cost depends on the number of users, then the placement of the component is irrelevant. However, some licensing constructs specify fees depending on the number of machines (either VMs or PMs) on which the software runs, in some cases also weighting the number of machines with some coefficients of computing power. For such licensing models, the placement of the components does matter: colocating multiple components with the same license on the same VM or PM leads to a reduction of the license fees to be paid.

A.4 A note on component sizing

In this work, we focus on software deployment. At this stage we can assume that the components and their sizes (i.e., resource requirements) are given. The components and their sizes must be determined in an earlier phase of the software development process in such a way that the performance of the application complies with the SLOs that it has to fulfill. This is a challenging task, but it is outside the scope of this paper.

1. Graphical Processing Unit

References

[1] H. Jin, H. Qin, S. Wu, and X. Guo, "CCAP: A cache contention-aware virtual machine placement approach for HPC cloud," *International Journal of Parallel Programming*, vol. 43, no. 3, pp. 403–420, 2015.

[2] A. Lefray, E. Caron, J. Rouzaud-Cornabas, and C. Toinard, "Microarchitecture-aware virtual machine placement under information leakage constraints," in *IEEE 8th International Conference on Cloud Computing*, 2015, pp. 588–595.

[3] X. Li, A. Ventresque, J. O. Iglesias, and J. Murphy, "Scalable correlation-aware virtual machine consolidation using two-phase clustering," in *International Conference on High Performance Computing & Simulation*, 2015, pp. 237–245.

[4] Z. A. Mann, "Interplay of virtual machine selection and virtual machine placement," in *Proceedings of the 5th European Conference on Service-Oriented and Cloud Computing*, 2016, pp. 137–151.

[5] Y. Ren, L. Liu, Q. Zhang, Q. Wu, J. Guan, J. Kong, H. Dai, and L. Shao, "Shared-memory optimizations for inter-virtual-machine communication," *ACM Computing Surveys*, vol. 48, no. 4, 2016.

[6] A. Zhou, S. Wang, Z. Zheng, C.-H. Hsu, M. R. Lyu, and F. Yang, "On cloud service reliability enhancement with optimal resource usage," *IEEE Transactions on Cloud Computing*, vol. 4, no. 4, pp. 452–466, 2016.