Optimized Cloud Deployment of Multi-tenant Software Considering Data Protection Concerns

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

Paper published in the Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2017), pages 609-618, IEEE Press, 2017

***Abstract*—Concerns about protecting personal data and intellectual property are major obstacles to the adoption of cloud services. To ensure that a cloud tenant's data cannot be accessed by malicious code from another tenant, critical software components of different tenants are traditionally deployed on separate physical machines. However, such physical separation limits hardware utilization, leading to cost overheads due to inefficient resource usage. Secure hardware enclaves offer mechanisms to protect code and data from potentially malicious code deployed on the same physical machine, thereby offering an alternative to physical separation. We show how secure hardware enclaves can be employed to address data protection concerns of cloud tenants, while optimizing hardware utilization. We provide a model, formalization and experimental evaluation of an efficient algorithmic approach to compute an optimized deployment of software components and virtual machines, taking into account data protection concerns and the availability of secure hardware enclaves. Our experimental results suggest that even if only a small percentage of the physical machines offer secure hardware enclaves, significant cost savings can be achieved.**

***Index Terms*—virtual machine placement; cloud deployment; data protection; privacy; secure computing**

I. Introduction

Ensuring the protection of critical business data (intellectual property) and sensitive personal data is key for business success and end-user adoption of cloud services [1]. Data protection concerns may especially arise in a multi-tenant setting, in which the confidentiality of a tenant's data may be breached by malicious code from another tenant.

Virtualization offers some level of data protection by deploying different tenants' code and data in different virtual machines (VMs). However, if these separate VMs are deployed on the same physical machine (PM), malicious code in one VM may still breach the confidentiality of data in another VM; e.g., by means of covert channels in the underlying hardware [2], [3]. To address such security risks, the traditional solution is to physically separate critical code and data of one tenant from the code and data of other tenants by deploying each on different PMs. However, physical separation reduces the opportunity for sharing resources, thus leading to limited hardware utilization and increased costs.

Secure enclaves (such as offered by intel's SGX technology[1]) provide hardware mechanisms to protect critical code and data, maintaining confidentiality even when an attacker has physical control of the hardware platform and can conduct direct attacks on memory [4]. Secure enclaves thereby make it possible to protect code and data within a PM, thus offering an alternative to physical separation.

Since PMs offering secure enclaves are likely to remain a scarce resource in data centers in the near future, a combination of secure hardware and physical separation on traditional hardware appears to be a good compromise to achieve data protection goals while aiming to optimize resource usage.

In recent years, resource-efficient cloud deployment has received much attention [5]. However, the problem we are addressing is much more difficult than traditional formulations. First, we have to take into account which software component contains code and data of which tenants and which of those data is critical for the tenant, also considering the possibility of multiple tenants sharing a component (multi-tenancy). Second, VMs need to be selected for the software components, sized, and placed on the PMs appropriately. Third, we have to consider a pool of PMs with different security attributes. A special challenge is that, since data protection requirements arise on the level of software components but security capabilities are given at the level of PMs, we need to address the deployment problem in a holistic way, from software components via VMs down to PMs. This is in contrast with previous work that focused either on deploying software components on VMs or VMs on PMs, but not both [6].

Thus, this paper aims at answering the following questions:

- How can data protection concerns be taken into account while optimizing resource usage of the deployment?
- Is it algorithmically feasible to solve a joint optimization problem with all the above aspects in acceptable time?
- Is it true that secure enclaves can be leveraged to improve PMs' utilization and thus reduce costs?

The paper answers these questions affirmatively, by making the following contributions:

- We define and formalize a **cloud deployment model considering data protection concerns**.
- We introduce **efficient heuristic algorithms to compute an optimized cloud deployment** for tenant components (i.e., code and data) and VMs, taking into account data

[1]Software Guard Extensions, see https://software.intel.com/en-us/sgx

protection concerns and capacity constraints.

- By means of a **comprehensive empiric evaluation with real workload data**, we analyze how cost savings depend on data security properties. In particular, we find that even if only 20% of the PMs offer secure hardware enclaves, savings of energy consumption (which is a major cost driver) may be as high as 47.5%.

After a discussion of related work in Section II, we introduce our cloud deployment model in Section III and its formalization in Section IV. Our algorithms are described in Section V, followed by a case study in Section VI and experimental evaluation in Section VII.

II. Related work

The *generic* problem of cloud deployment has received significant attention in the literature; e.g., see [7], [5]. Most solutions focus either on placing VMs on PMs [8], [9] or on selecting VMs for deploying software components [10], [11]. These existing solutions mainly consider performance, costs, and energy consumption, but do not explicitly take into account data protection concerns. Therefore, the resulting deployments may violate data protection requirements.

Cloud deployment *considering data protection concerns* has been approached from the point of view of cloud users and cloud providers. From the point of view of cloud users that aim to deploy cloud components on public or multi-clouds, Massonet et al. formalize security requirements (together with other quality requirements) of the cloud components and the security mechanisms offered by the cloud providers as a constraint programming problem [12]. The solution of the problem delivers an optimal selection of the cloud service providers for deploying the given cloud components. Garcia et al. focus on comparing cloud providers by assessing the level of security they provide [13]. These approaches do not consider specific characteristics of the concrete compute hardware (such as secure hardware enclaves), because internals of the cloud (in particular the underlying PMs) are hidden from the cloud user. In addition, these approaches consider the problem for a single cloud user and as such do not address multi-tenancy concerns.

From the point of view of cloud providers aiming to optimize resource allocation, Caron et al. propose a set of heuristics for mapping VMs to PMs, while considering security constraints [14], [2]. Their solution focuses on one particular security threat, which is the potential data leakage among VMs due to their placement on the same PM. To this end, cloud users may specify the level of isolation they require. Similarly, Shetty et al. address the placement of VMs on PMs with the aim of minimizing the impact of vulnerabilities of one VM on others that are hosted on the same PM [15]. However, these approaches only consider the mapping of VMs to PMs and do not concern themselves with multi-tenant software components and their respective, more fine-grained security concerns. They only focus on physical separation and do not take into account the possibility to deploy onto secure hardware, allowing even critical VMs to be deployed on the same PM.

III. Cloud deployment model

We now describe our cloud deployment model considering data protection concerns. A diagrammatic representation of the model with its key concepts is shown in Fig. 1. The model takes the point of view of a cloud provider that uses its own PMs to offer Software-as-a-Service (SaaS) or Platform-as-a-Service (PaaS) solutions to various **Tenants**.

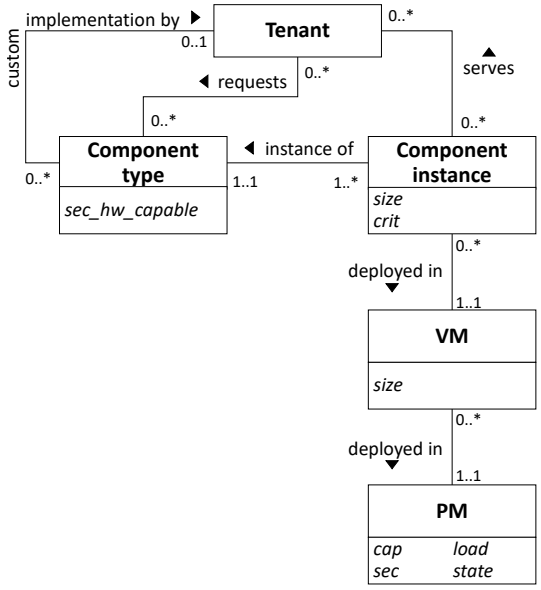


Fig. 1. Cloud deployment model considering data protection concerns

The cloud services are implemented by **Components** that are hosted in the provider's virtualized data center. Components may contain code, data or both. To offer tenants customized services, there can be multiple instances of the same component, so that different tenants may use different component instances [16]. Hence, we differentiate between **Component Types** and **Component Instances**. A tenant may *request* a set of component types. The provider has to decide for each requested component type which instance of the given type should *serve* the given tenant, taking into account the possibilities of reusing existing component instances. This is also where data protection concerns are formulated: for each requested component type, the tenant specifies whether the given component is critical (*crit*) in terms of data protection. If so, the component instance of the given type serving the tenant will be handled by the provider accordingly. In particular, the component instance will be a dedicated instance and not shared with other tenants. Note that the same component type can be critical for one tenant and non-critical for another tenant. This is why *crit* is an attribute of the component instance and not of the component type.

In the case of SaaS, beyond the component types implemented by the cloud provider, tenants may create their own implementation of a component type or customize an existing component type. In the case of PaaS, usually most component types are implemented by the tenants. Therefore, in our model a component type may be implemented by either the provider

or one of the tenants; the latter case is modeled through the *custom implementation by* relation. We assume that tenants trust the provider but not each other, i.e., components created by other tenants may pose a threat.

Actual deployment happens at two levels: component instances are *deployed in* **VMs** and VMs are *deployed in* **PMs**. Each PM may offer certain capacities (*cap*) for its various resource types, such as CPU, memory, and disk. Component instances and VMs have resource requirements according to these resource types, which we call their *size*. The size of a component instance depends on the tenants using it. The size of a VM arises from the size of the component instances that it hosts. The *load* of a PM is the aggregated size of the VMs it hosts.

To serve the tenant requests, components have to be instantiated by the cloud provider, deployed on VMs, which must be deployed on PMs. To address data protection requirements, our model facilitates two mechanisms to protect sensitive tenant data from untrusted code of other tenants:

- *Physical separation.* As mentioned in the introduction, this is the traditional approach to ensure that malicious code in one VM may not breach the confidentiality of data in a VM co-located on the same PM.
- *Secure enclaves.* Some PMs may support secure enclaves, in which data processing takes place in a protected environment (e.g., intel SGX). On such a PM, a VM hosting a critical component of one tenant may be co-located with another VM hosting the non-trusted component of another tenant, as long as the critical component is run in a secure enclave[2]. The *sec* attribute of the PM encodes whether the PM supports secure enclaves. A prerequisite of secure sharing is also that the critical component be able to take advantage of secure enclaves (*sec_hw_capable*)[3].

Previous work focused only on parts of Fig. 1, e.g., the upper half (multi-tenant software provisioning [16]) or the lower half (VM provisioning in virtualized data centers [5]). However, we argue that the effective handling of data protection concerns requires an end-to-end approach, since data protection requirements arise at the level of component types, but data protection mechanisms reside at the PM level.

IV. Deployment model formalization

Based on the cloud deployment model from Section III, this section formally describes the constraints that a cloud deployment needs to maintain.

As Table I shows, let X denote the set of tenants. The set of component types provided by the provider itself is denoted by C_{std}. The set of custom component types created by tenant $x \in X$ is denoted by $C_{cust}(x)$. Let

$$C := C_{std} \cup \bigcup_{x \in X} C_{cust}(x)$$

[2]For virtualization on SGX-enabled hardware, see https://01.org/intel-software-guard-extensions/sgx-virtualization.

[3]To leverage SGX, a program needs to explicitly create an enclave, add code and/or data to it etc., using the special instruction set of SGX.

TABLE I
Summary of notation

Notation	Explanation
X	Set of tenants
C	Set of all component types
$C(x)$	Set of component types requested by tenant x
C_{std}	Set of standard component types
$C_{cust}(x)$	Set of custom component types created by tenant x
I	Set of all deployed component instances
$I(c)$	Set of instances of component type c
$c(i)$	Component type of component instance i
$X(i)$	Set of tenants served by component instance i
V	Set of VMs
$v(i)$	VM hosting component instance i
d	Number of resource types
$size(i)$	Size of component instance i
$\Delta(i, x)$	Size increase of component instance i due to tenant x
$size(v)$	Size of VM v
s_0	Size of an empty VM
P	Set of PMs
$cap(p)$	Capacity of PM p
$p(v)$	PM hosting VM v
$load(p)$	Total size of the VMs hosted by PM p

denote the set of all component types.

For each component type $c \in C$, the set of currently deployed instances of the given type is denoted by $I(c)$. Further, let $I := \bigcup_{c \in C} I(c)$ denote the set of all deployed component instances. The component type that component instance $i \in I$ belongs to is denoted by $c(i)$. For a component instance $i \in I$, the set of tenants that are served by i is denoted by $X(i) \subseteq X$. For a tenant x, the set of component types that the tenant requires is denoted by $C(x) \subseteq C$.

If tenant x requested a component of type c, there must be a component instance of the given type serving tenant x. This is expressed by the following constraint:

$$\forall x \in X, \forall c \in C(x) : \exists i \in I(c), x \in X(i). \tag{1}$$

The set of VMs currently in use is denoted by V. For each component instance $i \in I$, $v(i) \in V$ denotes the VM in which it is deployed.

The set of resource types (e.g., CPU, memory, disk) is denoted by R, the number of resource types is $d = |R|$. We formalize the size of a component instance i as a d-dimensional vector $size(i)$. The size of a component instance typically depends on the number of tenants served by the component instance and the load with which they use it. We formalize this as follows: the addition of a tenant x to component instance i leads to an increase of $size(i)$ by $\Delta(i, x)$. The size of a VM v is also a d-dimensional vector: the sum of the sizes of the component instances deployed in v, plus the overhead of virtualization:

$$size(v) = s_0 + \sum_{i \in I : v(i) = v} size(i),$$

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

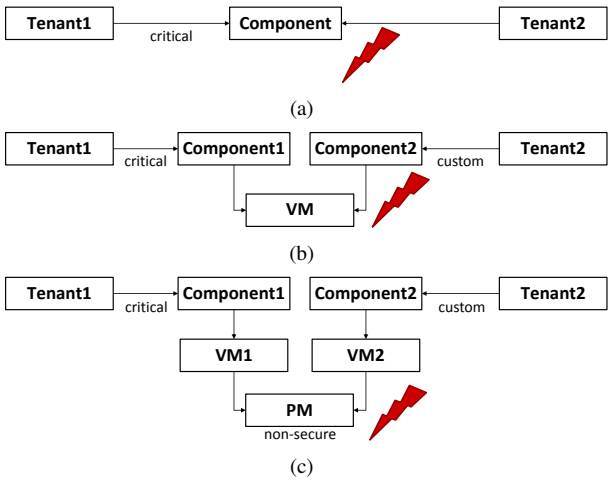


Fig. 2. Potential violations of data security requirements

The set of available PMs is denoted by P. 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 $cap(p)$. For a VM v, the PM that hosts the VM is denoted by $p(v)$. The mapping of VMs on PMs must respect the capacity of the PMs:

$$\forall p \in P: \quad load(p) = \sum_{v \in V : p(v) = p} size(v) \leq cap(p). \tag{2}$$

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 $j = 1, \ldots, d$ [17].

Further constraints arise from data protection requirements. A critical component instance must not be shared by more than one tenant (Fig. 2(a)):

$$\forall i \in I : (crit(i) \Rightarrow |X(i)| = 1). \tag{3}$$

A component marked as critical by a tenant must not be in the same VM as custom components of other tenants, so that data protection violation by malicious code running within the same VM can be avoided (Fig. 2(b)):

$$\forall i, i' \in I : (crit(i), X(i) = \{x\}, v(i) = v(i') \Rightarrow c(i') \in C_{std} \cup C_{cust}(x)). \tag{4}$$

If a VM accommodating a critical component instance i of a tenant is deployed on PM p, then either p must support secure enclaves and i must be capable of taking advantage of secure enclaves, or p must not host any VM with a custom component instance of another tenant (Fig. 2(c)):

$$\forall i, i' \in I : (crit(i), X(i) = \{x\}, p(v(i)) = p(v(i')) \Rightarrow (sec(p(v(i))) \wedge sec\_hw\_capable(c(i))) \vee c(i') \in C_{std} \cup C_{cust}(x)). \tag{5}$$

Beside complying with constraints (1)-(5), our aim is to minimize overall energy consumption because of its impact on operational costs and the environment. We assume that the power consumption of a PM is 0 when switched off, otherwise it is given by a function depending on the PM's CPU load.

V. Algorithmic Approach

Our algorithmic approach for computing optimized deployment automates the following three main types of decisions to be made by a SaaS/PaaS provider:

- Creation and removal of component instances, and their mapping to tenants (top layer of Fig. 1)
- Mapping of components to VMs (middle layer of Fig. 1)
- Mapping of VMs to PMs (bottom layer of Fig. 1)

Decision-making may happen in both event-triggered and time-triggered manner. Events requiring immediate reaction include component instantiation requests from new tenants and termination requests from existing tenants. On the other hand, the provider may periodically re-optimize mappings to react to relevant changes in the workload [9].

Next, we describe our proposed algorithms for handling new requests, handling termination requests, and performing re-optimization. Each algorithm must strike a balance between the objectives of data protection and cost minimization. Because of the hardness of the problem [18], we use heuristics.

A. Handling of a new request

A request r is given by the tuple $(x_r, c_r, crit_r)$, where x_r is a tenant, c_r is a component type, and the flag $crit_r$ specifies the criticality of the component instance for the tenant.

Algorithm 1 Handling a new request

```
1:  procedure PROCESS_REQUEST(x_r, c_r, crit_r)
2:      if ∃i ∈ I(c_r) s.t. MAY_INST_HOST_TENANT(i, x_r, crit_r) then
3:          map x_r to i
4:      else
5:          let i_new be a new instance of c_r
6:          map x_r to i_new
7:          if ∃v ∈ V s.t. MAY_VM_HOST_INSTANCE(v, i_new) then
8:              map i_new to v
9:          else
10:             let v_new be a new VM
11:             map i_new to v_new
12:             P' = SORT_PMS(P)
13:             if ∃p ∈ P' s.t. MAY_PM_HOST_VM(p, v_new) then
14:                 let p_0 be the first such PM in P'
15:                 if p_0 is off then
16:                     switch on p_0
17:                 end if
18:                 map v_new to p_0
19:             else
20:                 return failure
21:             end if
22:         end if
23:     end if
24:     return success
25: end procedure
```

As shown in Algorithm 1, we first aim to reuse an existing component instance to accommodate the new request (lines 2-3) and create a new component instance only if such reuse is not possible (lines 4-6). In the latter case, the newly created component instance needs to be placed on a VM. Again, the algorithm first tries to reuse an existing VM for this purpose

Algorithm 2 Subroutines to determine placeability

```
1: procedure MAY_INST_HOST_TENANT(i, x_r, crit_r)
2:     if load(p(v(i))) + Δ(i, x) ≰ cap(p(v(i))) then
3:         return false
4:     end if
5:     if (¬crit(i) ∧ ¬crit_r) ∨ X(i) ⊆ {x_r} then
6:         return true
7:     else
8:         return false
9:     end if
10: end procedure
11:
12: procedure MAY_VM_HOST_INSTANCE(v, i_new)
13:     if load(p(v)) + size(i_new) ≰ cap(p(v)) then
14:         return false
15:     end if
16:     if crit(i_new) ∧ X(i_new) = {x} ∧ ∃i (v(i) = v ∧ c(i) ∉
        C_std ∪ C_cust(x)) then
17:         return false
18:     else if ∃i (v(i) = v ∧ crit(i) ∧ X(i) = {x} ∧ c(i_new) ∉
        C_std ∪ C_cust(x)) then
19:         return false
20:     else
21:         return true
22:     end if
23: end procedure
24:
25: procedure MAY_PM_HOST_VM(p, v_new)
26:     if load(p) + size(v_new) ≰ cap(p) then
27:         return false
28:     end if
29:     if ∃i, i' (((v(i) = v_new ∧ p(v(i')) = p) ∨ (v(i') =
        v_new ∧ p(v(i)) = p)) ∧ crit(i) ∧ X(i) = {x} ∧ c(i') ∉
        C_std ∪ C_cust(x)) ∧ ¬(sec(p) ∧ sec_hw_capable(c(i))) then
30:         return false
31:     else
32:         return true
33:     end if
34: end procedure
```

(lines 7-8) and creates a new VM only if none of the existing VMs can host the new component instance (lines 9-11). If a new VM was created, it is placed on a PM (lines 12-18). The algorithm's attempts to reuse existing component instances and VMs help to avoid unnecessary costs.

The questions whether an existing component instance can host one more tenant, whether an existing VM can host a new component instance, and whether a PM can host a new VM are answered by the appropriate subroutines shown in Algorithm 2. Each subroutine investigates the implications with respect to the capacity constraints and data protection constraints, in line with the cases shown in Fig. 2. Even though the three subroutines are similar, they differ in important details.

In MAY_INST_HOST_TENANT, the algorithm must make sure that the load of the PM accommodating the given component instance does not grow too large when the component instance grows as a result of serving one more tenant (lines 2-4). Note that "$\not\leq$" between vectors means that in at least one dimension the left side is greater than the right side. Moreover, we must make sure not to share a critical component between different tenants (lines 5-9).

In MAY_VM_HOST_INSTANCE, the algorithm must make sure that the load of the PM hosting the given VM does not grow too large because of the new component instance (lines 13-15). If the new component instance is a critical component dedicated to tenant x, then there must be no custom components created by another tenant in the VM (lines 16-17), and vice versa, if there is a critical component instance in the VM, then the new component instance must not be a custom component instance of a different tenant (lines 18-19).

Finally, MAY_PM_HOST_VM ensures that the aggregate size of the VMs remains below the capacity of the PM (lines 26-28). Furthermore, it is checked whether there is a component instance in the VM and another in the PM or vice versa that would violate the data protection constraint, taking into account the criticality and custom nature of the components, as well as the security capabilities of the PM and whether the critical component could take advantage of such capabilities (lines 29-33).

One more detail to be clarified for Algorithm 1 is the order in which the algorithm searches for a PM (in line 13, based on the SORT_PMS call in line 12). To minimize energy consumption, a new PM should be turned on only if necessary. Hence, the algorithm orders the PMs based on their power state: PMs that are turned on precede those that are turned off. Since we assume that secure PMs are a scarce resource, the algorithm aims to use non-secure PMs whenever possible. Therefore, within the two PM groups based on power state, we sort PMs such that non-secure ones precede secure ones, leading to the following partial order (x and y are two PMs):

$$x \prec y \Leftrightarrow (state(x) = on \wedge state(y) = off) \\ \vee (state(x) = state(y) \wedge \neg sec(x) \wedge sec(y)).$$

B. Termination of a request

Algorithm 3 Termination of a request

```
1: procedure TERMINATE_REQUEST(x_r, i_r)
2:     remove x_r from X(i_r)
3:     if X(i_r) = ∅ then
4:         remove i_r from v(i_r)
5:         if v(i_r) becomes empty then
6:             remove v(i_r) from p(v(i_r))
7:             if p(v(i_r)) becomes empty then
8:                 switch off p(v(i_r))
9:             end if
10:        end if
11:    end if
12: end procedure
```

The termination of a request r means that a tenant x_r stops using a component instance i_r. As shown in Algorithm 3, removing x_r from $X(i_r)$ may result in $X(i_r)$ becoming empty, in which case it makes sense to remove the component instance to avoid unnecessary resource consumption. Removing i_r may in turn lead to its accommodating VM becoming empty, in which case it is again useful to remove the entire VM. If this way the accommodating PM also becomes empty, then it can be switched off.

C. Re-optimization

The above algorithms for handling new requests and termination requests make local decisions and minimal modifications to react quickly to external events. In the long run, such greedy choices can lead to sub-optimal solutions and thus to unnecessarily high operational costs[4]. This is why it is useful to check time and again if the operation of the system can be optimized by means of live migration of VMs[5].

Algorithm 4 Re-optimization

```
1:  procedure RE-OPTIMIZE
2:      // check if an active PM can be emptied
3:      for all p ∈ P with state(p) = on do
4:          for all v ∈ V with p(v) = p do
5:              for all p' ∈ P with state(p') = on do
6:                  if MAY_PM_HOST_VM(p', v) then
7:                      tentatively migrate v from p to p' and go to
    next VM
8:                  end if
9:              end for
10:         end for
11:         if p has become empty then
12:             commit the tentative migrations
13:             switch off p
14:         else
15:             undo the tentative migrations
16:         end if
17:     end for
18:     // check if a secure PM can take the load from two (non-
    secure) PMs
19:     while ∃p, p1, p2 ∈ P : sec(p), state(p) = off, state(p1) = state(p2) = on, load(p1) + load(p2) ≤ cap(p) do
20:         switch on p
21:         migrate all VMs from p1 and p2 to p
22:         switch off p1 and p2
23:     end while
24:     // check if the load of a secure PM can be moved to a non-
    secure PM
25:     while ∃p, p' ∈ P : sec(p), state(p) = on, ¬sec(p'), state(p') = off, ∄i, i' : (p(v(i)) = p(v(i')) = p, crit(i), X(i) = {x}, c(i') ∉ C_std ∪ C_cust(x)) do
26:         switch on p'
27:         migrate all VMs from p to p'
28:         switch off p
29:     end while
30: end procedure
```

As shown in Algorithm 4, three kinds of optimization opportunities are explored. The first is the traditional consolidation [19]: emptying an active PM by migrating all its VMs to some other already active PMs (lines 2-17). The subroutine MAY_PM_HOST_VM is reused here to make sure that the capacity and data protection constraints are not violated by the migrations.

The second optimization opportunity arises if there is a pair of PMs, the loads of which would allow consolidating them to a single PM, but this is not allowed because the two PMs are non-secure and host VMs that need to be separated because of data protection reasons. In this case, the traditional consolidation step cannot be applied. But if a secure PM can be switched on, the load of the two non-secure PMs can be migrated to the secure one, and the two emptied PMs can be switched off (lines 18-23), thus ultimately decreasing the number of active PMs by one.

The third optimization opportunity does not decrease the number of active PMs but fosters economic handling of secure PMs as scarce resources: if there is an active secure PM, the VMs of which would not need separation, then they can all be migrated to a newly switched-on non-secure PM, and the secure PM can be switched off (lines 24-29).

D. Run-time complexity of the algorithms

TABLE II
ASYMPTOTIC EXECUTION TIME OF THE ALGORITHMS

Algorithm	Worst-case execution time
PROCESS_REQUEST	$O(\|I\|+\|V\|\cdot I_{\max}+\|P\|\cdot\log\|P\|+\|P\|\cdot I_{\max}^2)$
TERMINATE_REQUEST	$O(1)$
RE-OPTIMIZE	$O(\|P\|^2\cdot V_{\max}\cdot I_{\max}^2+\|P\|^3\cdot V_{\max}+\|P\|^2\cdot(I_{\max}^2+V_{\max}))$

An analysis of the presented algorithms reveals the asymptotic bounds on their run-time complexity as shown in Table II. Here, $I_{\max}$ is an upper bound to the number of component instances within a single VM or PM, whereas $V_{\max}$ is an upper bound to the number of VMs in a single PM; both of these bounds are typically not too large. As can be seen, all three algorithms have polynomial complexity and thus exhibit efficient execution times, with RE-OPTIMIZE having the highest complexity in line with its more global coverage.

These algorithms all are heuristics with no guarantee of optimality regarding optimization criteria like energy consumption or the number of active PMs [18]. However, the algorithms do guarantee that all constraints – both data protection constraints and capacity constraints – are always obeyed.

VI. CASE STUDY

We implemented our algorithms in C++ and tested them in a simulation environment.[6] The program is publicly available from https://sourceforge.net/p/vm-alloc/multitenant/.

To demonstrate the applicability and effectiveness of our optimization approach, we employ the cloud-based variant of the CoCoME case study [20]. CoCoME models cloud services that support the typical trading operations of a supermarket chain, like the management of stores, inventory management, and product dispatching. As such, CoCoME as SaaS offers a realistic case study covering both efficiency concerns of cloud data centres and data protection concerns of tenants.

An example deployment – created by our algorithm – for 3 tenants (A, B, C) is shown in Fig. 3a. As can be seen, non-critical standard components like the **ProductDispatcher** can

[4]But data protection requirements are always satisfied.

[5]Although some components might support migration between VMs, this cannot be assumed in general, so we do not rely on such mechanisms.

[6]In contrast to existing cloud simulators which do not account for critical or tenant-specific components, nor secure hardware, our program was explicitly written to support all concepts of our problem formulation.

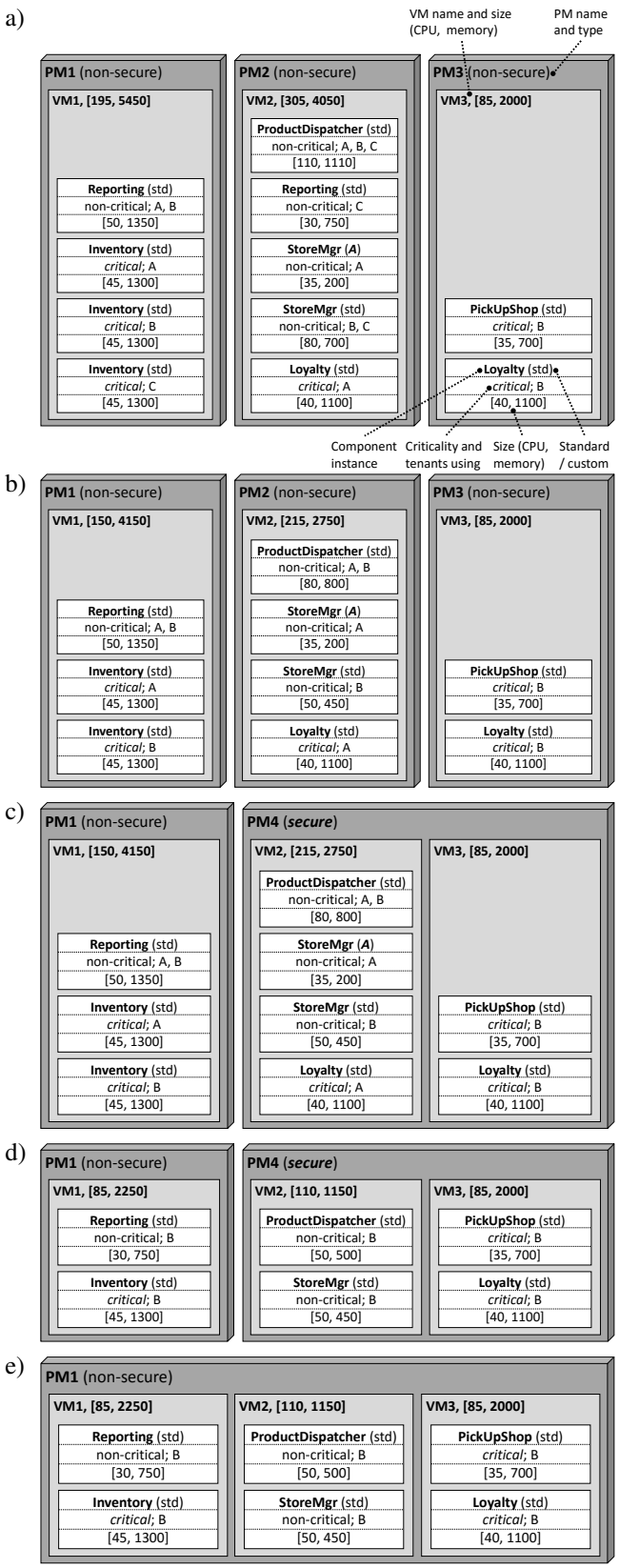


Fig. 3. Sample CoCoME scenario. a) Starting state with tenants A, B, C. b) State after tenant C left. c) State after re-optimization. d) State after tenant A left. e) State after further re-optimization. Each PM has capacity [400, 6000].

be shared by multiple tenants. Some components are critical because they contain personally identifiable information, such as the **Loyalty** component that offers rebates based on personal purchase history, or they may contain business secrets, such as the **Inventory** component. These components are not shared. Moreover, tenant A implemented their own **StoreMgr** component. Since this component may contain malicious code, the critical components of the other tenants are not on the same PM as the **StoreMgr** of tenant A.

Now assume that tenant C terminates its contract with the cloud provider. Fig. 3b shows the resulting system state after our algorithm processed the termination request. Now VM2 and VM3 are small enough so that they could be consolidated to a single PM. However, this would violate the data protection constraint since A's custom component and B's critical component would be on the same PM. Our algorithm solves this problem by turning on a new, secure PM, migrating VM2 and VM3 to this PM, and switching off their old PMs (Fig. 3c). If subsequently also tenant A leaves, this results in the configuration depicted in Fig. 3d. Now the algorithm can consolidate all VMs to a single non-secure PM (Fig. 3e).

As can be observed, all data protection requirements are fulfilled throughout the scenario, while the number of used PMs is always minimized.

VII. Evaluation

To assess the performance of the algorithms and the dependence of the results on different problem parameters, we performed a set of controlled experiments with real-world test data modeling a PaaS provider.

A. Experiment setup

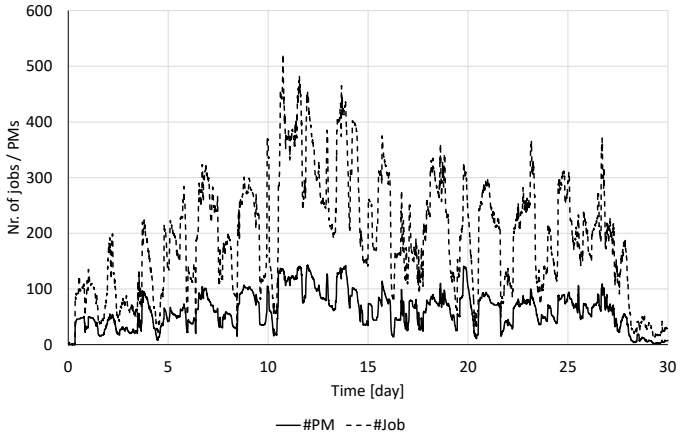
For the components, we used a real workload trace from the Grid Workloads Archive, namely the AuverGrid trace[7]. From the trace, we used the first n_{job} jobs (where n_{job} varied from 10,000 to 60,000) that had valid CPU and memory usage data. The simulated time (i.e., the time between the start of the first job and the end of the last one) was one month, thus giving sufficient exposure to practical workload patterns. Each job was mapped to a request where the size of the job from the trace was used as component size increase $\Delta(i, x)$ for the request. The finishing of a job was mapped to an appropriate termination request.

Since the workload trace does not contain all information we need, we generated the missing information as follows:

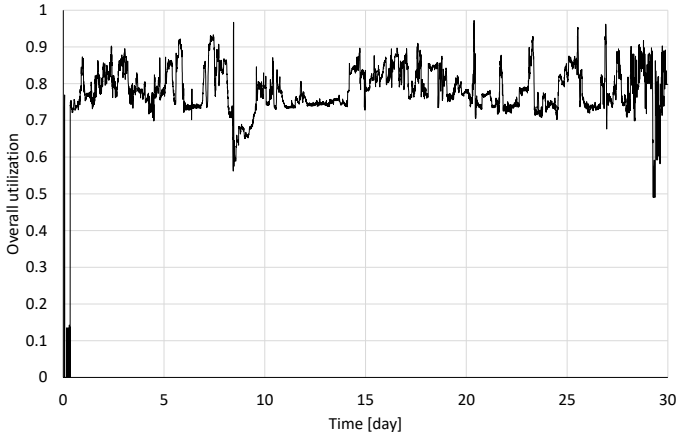
- Each job was marked as critical with probability p_{crit}.
- Each job was marked as custom with probability p_{cust}.
- Each job was marked as capable of using secure enclaves with probability p_{cap}.
- We generated a number n_{ten} of tenants, and assigned each job randomly to one of the tenants.

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

[7]Available from http://gwa.ewi.tudelft.nl/datasets/gwa-t-4-auvergrid



(a) Number of jobs and number of PMs



(b) Overall utilization of the PMs by the jobs

Fig. 4. Time series of an example simulation run ($p_{sec} = p_{crit} = p_{cust} = p_{cap} = 1$, $n_{job} = 30000$, $n_{ten} = 7500$)

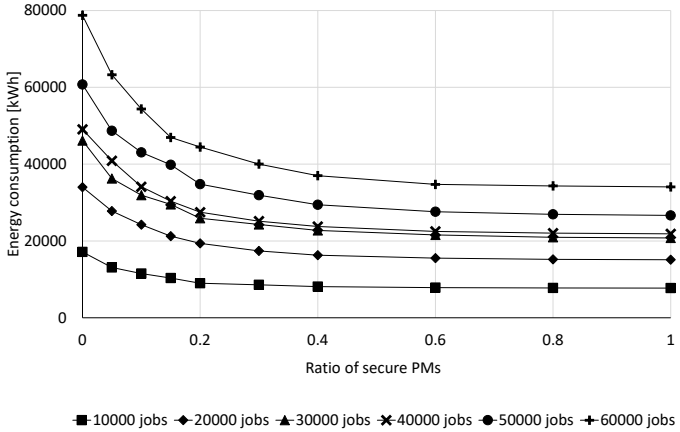


Fig. 5. Energy consumption as a function of the number of jobs and the ratio of secure PMs ($p_{crit} = p_{cust} = p_{cap} = 1$, $n_{ten} = n_{job}/4$)

540W (full load) [21]. Each PM was marked as secure with a probability of p_{sec}. 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 [22] and 107-566 MB for memory [23]. In our experiments, we use 5% CPU overhead and 200 MB memory overhead. The VM placement is re-optimized every 5 minutes, like in [24]. Several metrics are logged, including energy consumption, number of active jobs, number of active PMs, utilization of the PMs, and the number of migrations. We performed measurements on a Lenovo ThinkPad X1 laptop with Intel Core i5-4210U CPU @ 1.70GHz and 8GB RAM.

B. Time series analysis

Fig. 4(a) shows the temporal development of the number of active jobs and the number of active PMs in an example simulation run. As can be observed, the capacity of the system (measured by the number of active PMs) closely follows the demand (the number of active jobs). The capacity also reacts quickly to sudden changes in the demand. As a result, the overall utilization of the system is continuously very high (except for the beginning and end of the simulation, when the number of jobs and PMs is low), as documented by Fig. 4(b). Overall utilization is computed as $\max(cpu\_utilization, memory\_utilization)$, where $cpu\_utilization$ is the total CPU demand of all active jobs divided by the total CPU capacity of all active PMs and $memory\_utilization$ is computed analogously. Note that this is actually lower than the average physical utilization of the PMs, because the latter also includes the virtualization overhead that we do not include in our metric.

C. Costs

Next, we investigate how different parameter settings impact energy consumption as a key cost metric. Fig. 5 shows the dependence of energy consumption on n_{job} and p_{sec}. The figure reinforces our hypothesis that secure PMs offer more possibilities for consolidating the workload, which in turn leads to energy and thus also cost savings. As can be seen in the figure, the savings can be substantial. For 10,000 jobs, 20% secure PMs lead to 47.5% reduction in energy consumption. Further increasing the ratio of secure PMs leads to even higher reduction in energy consumption, but obviously with a diminishing returns pattern. At the extreme, having 100% secure PMs leads to a reduction in energy consumption of an additional 7.4% over the 20% case. The curves of the figure representing higher numbers of jobs exhibit the same trend. Some slight changes can be observed though: the reduction of energy consumption from $p_{sec} = 0\%$ to $p_{sec} = 20\%$ gets a bit less (e.g., for $n_{job} = 60000$, it is 43.5%), but the reduction from $p_{sec} = 20\%$ to $p_{sec} = 100\%$ gets higher (for $n_{job} = 60000$, it is 13.2%), so that the overall reduction from $p_{sec} = 0\%$ to $p_{sec} = 100\%$ also gets higher (56.7% for $n_{job} = 60000$ versus 54.9% for $n_{job} = 10000$).

We also experimented with varying the ratio of critical components, the ratio of custom components, and the ratio of components capable of using secure enclaves. Those experiments lead to very similar plots, which are therefore skipped.

The effect of the number of tenants for a constant number of requests is shown in Fig. 6. If the number of tenants is

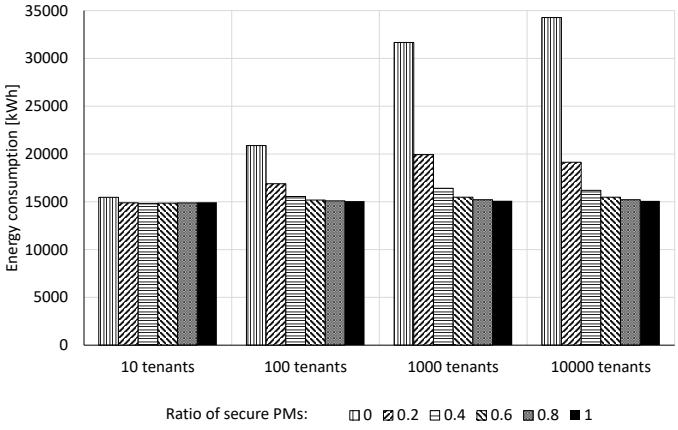


Fig. 6. Energy consumption as a function of the number of tenants and the ratio of secure PMs ($p_{crit} = p_{cust} = p_{cap} = 1$, $n_{job} = 20000$)

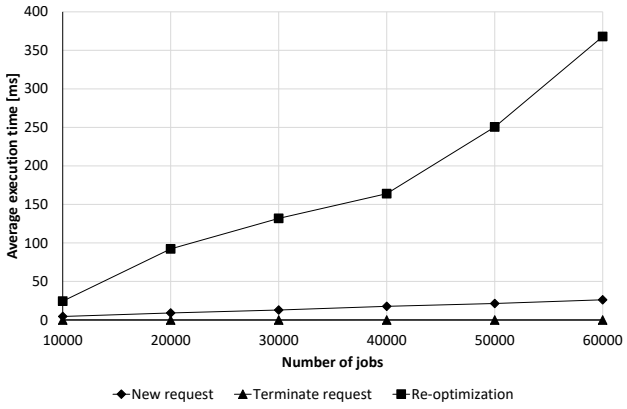


Fig. 7. Execution time of the algorithms as a function of the number of jobs ($p_{crit} = p_{cust} = p_{cap} = 1$, $n_{ten} = n_{job}/4$)

low, then each tenant has a large number of jobs. Since the jobs of the same tenant can be placed on the same VM or PM without any restrictions, the statistical multiplexing effect [25] leads to very good utilization within the set of jobs of each tenant, even if there are no secure PMs. In other words, the addition of secure PMs hardly helps to increase utilization, which is the reason why the energy consumption is hardly affected by the ratio of secure PMs. If, however, the number of tenants is high, then the average number of jobs per tenant is low. In this case, in the absence of secure PMs, consolidation opportunities are rather limited (only within the small sets of jobs of the same tenant). Hence, the emergence of secure PMs creates many new consolidation opportunities, which leads to a significant reduction in energy consumption.

D. Scalability

Fig. 7 shows how the execution time of the proposed algorithms scales with increasing problem size. In accordance with the results of Section V-D, we find that the execution time of both PROCESS_REQUEST and TERMINATE_REQUEST are negligible. The execution time of RE-OPTIMIZE is of course much higher. However, even for 60,000 jobs and 15,000 tenants, where up to 700 active PMs are used in parallel, the execution time of RE-OPTIMIZE stays below 0.4 seconds. Thus it can be efficiently used with practical problem sizes.

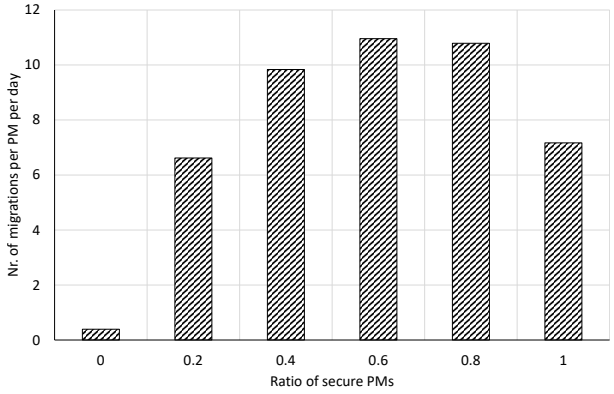


Fig. 8. Number of migrations per active PM per day, as a function of the ratio of secure PMs ($p_{crit} = p_{cust} = p_{cap} = 1$, $n_{job} = 20000$ $n_{ten} = 100$)

E. Migrations

Finally, we analyze the number of migrations caused by the RE-OPTIMIZE algorithm. This is important because too many migrations could lead to performance degradation or could even make the system unstable [26]. As shown in Fig. 8, the average number of migrations per active PM is less than 12 per day. This is a reassuringly low number that does not threaten the performance or stability of the system.

The pattern shown by Fig. 8 also gives some interesting insight. When there are no secure PMs, there are very few consolidation opportunities, so that also the number of migrations is very low. As secure PMs emerge, they lead to a significant increase in the number of consolidation opportunities (the positive effects of which we have already seen in previous plots), which in turn results in more migrations. When the ratio of secure PMs is already high, adding even more secure PMs does not lead to more consolidation opportunities, so we could expect a plateau in the number of migrations. As can be seen in Fig. 8, there is a decline in migrations instead. This is probably due to a secondary effect: when the number of secure PMs is high, most jobs are already placed by PROCESS_REQUEST on a secure PM that they share with other tenants' components, so that RE-OPTIMIZE will rarely find a situation where the components from two non-secure PMs can be unified on a secure PM by means of migrations.

VIII. Conclusions and Future Work

We demonstrated that it is feasible to express data protection-aware deployment of cloud services as a single optimization problem. This problem considered a multi-tenant virtualized cloud system from software components down to the physical infrastructure, taking into account capacity constraints, data protection requirements, and the availability of secure hardware. Based on this problem, we introduced appropriate heuristics that allow efficiently carrying out component instantiation and deployments in an optimized way.

The SaaS case study and the empiric evaluation on a real-world workload led to the following observations:

- The suggested approach managed to optimize utilization while satisfying the data protection requirements.
- Resource usage followed the demand closely, leading to continuously high overall utilization.
- With 20% of the PMs offering secure enclaves, energy consumption could be reduced by up to 47.5%.
- The relative cost reduction is especially high if the average number of components per tenant is not too large.
- The proposed algorithms are very fast, with execution time below 0.4 second in each tested case.
- The number of migrations generated by the proposed approach is below 12 migrations per PM per day.

Based on these results, the proposed approach is appropriate for practical use, offering reduced costs for cloud providers and reduced risks for cloud tenants.

Our future work will include the extension of this work with other security mechanisms. Furthermore, we plan to evaluate our methods in a more realistic environment, i.e., within an existing cloud simulator and/or a real deployment.

While this paper focused on the theoretical possibilities and the expected benefits of secure enclave based data protection assurance, it is obviously still a long way until the practical implementation of such a scheme. Important technical challenges include, e.g., the definition of appropriate interfaces for tenants to specify their data protection requirement and the development of virtualization middleware capable of exploiting secure hardware enclaves.

Acknowledgments

This work received funding from the European Community's 7th Framework Programme (FP7/2007-2013) under grant 610802 (CloudWave), the European Union's Horizon 2020 research and innovation programme under grant 731678 (RestAssured), and the German Research Foundation under Priority Programme SPP1593: Design For Future - Managed Software Evolution, grant PO 607/3-2 (iObserve).

References

[1] Networked European Software and Services Initiative, "Security and privacy: From the perspective of software, services, cloud and data," http://www.nessi-europe.eu/Files/Private/NESSI_Security_Privacy_White_Paper_issue_1.pdf, 2016.

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

[3] C. Modi, D. Patel, B. Borisaniya, A. Patel, and M. Rajarajan, "A survey on security issues and solutions at different layers of cloud computing," *The Journal of Supercomputing*, vol. 63, no. 2, pp. 561–592, 2013.

[4] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[5] 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.

[6] ——, "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.

[7] F. L. Pires and B. Baran, "A virtual machine placement taxonomy," in *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2015, pp. 159–168.

[8] 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*, 2015.

[9] 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*, 2015, pp. 387–396.

[10] W. Li, P. Svärd, J. Tordsson, and E. Elmroth, "Cost-optimal cloud service placement under dynamic pricing schemes," in *Proceedings of the 6th IEEE/ACM International Conference on Utility and Cloud Computing*, 2013, pp. 187–194.

[11] 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.

[12] P. Massonet, J. Luna, A. Pannetrat, and R. Trapero, "Idea: Optimising multi-cloud deployments with security controls as constraints," in *Engineering Secure Software and Systems - 7th International Symposium*. Springer, 2015, pp. 102–110.

[13] J. L. Garcia, T. Vateva-Gurova, N. Suri, M. Rak, and L. Liccardo, "Negotiating and brokering cloud resources based on security level agreements," in *Proceedings of the 3rd International Conference on Cloud Computing and Services Science*, 2013, pp. 533–541.

[14] E. Caron and J. Rouzaud-Cornabas, "Improving users' isolation in IaaS: Virtual machine placement with security constraints," in *IEEE 7th International Conference on Cloud Computing*, 2014, pp. 64–71.

[15] S. Shetty, X. Yuchi, and M. Song, "Security-aware virtual machine placement in cloud data center," in *Moving Target Defense for Distributed Systems*. Springer, 2016, pp. 13–24.

[16] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl, "Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications," in *Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems (PESOS'09)*, 2009, pp. 18–25.

[17] 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.

[18] Z. A. Mann, "Approximability of virtual machine allocation: much harder than bin packing," in *Proc. 9th Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, 2015, pp. 21–30.

[19] ——, "Rigorous results on the effectiveness of some heuristics for the consolidation of virtual machines in a cloud data center," *Future Generation Computer Systems*, vol. 51, pp. 1–6, 2015.

[20] R. Heinrich, K. Rostami, and R. Reussner, "The CoCoME platform for collaborative empirical research on information system evolution," Karlsruhe Reports in Informatics, Tech. Rep., 2016.

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

[22] 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.

[23] 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.

[24] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, "An integrated approach to resource pool management: Policies, efficiency and quality metrics," in *IEEE International Conference on Dependable Systems and Networks*, 2008, pp. 326–335.

[25] Y. Tan, F. Wu, Q. Wu, and X. Liao, "Resource stealing: a resource multiplexing method for mix workloads in cloud system," *The Journal of Supercomputing*, pp. doi:10.1007/s11 227–015–1609–3, 2016.

[26] 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.