



Intelligent and compliant dynamic software license consolidation in cloud environment

Leila Helali¹ · Mohamed Nazih Omri¹

Received: 1 November 2021 / Accepted: 5 July 2022 / Published online: 25 July 2022

© The Author(s), under exclusive licence to Springer-Verlag GmbH Austria, part of Springer Nature 2022

Abstract

Based on the virtualization technology and pushed by the softwarization paradigm and the actual demand for services and resources, commercial cloud data centers know an unprecedented expansion. The systematic presence of software and the services generated enabled the development of the already dense application expenses. This causes, not only a cost explosion, especially when proprietary solutions protected by licenses are in hand, but also, represents a critical need in terms of software asset and resource management at the SaaS level. In addition to these costs, inefficient resource utilization, and the resulting energy represent an important part of the operational expenditure of data centers and are still a hot topic despite the consolidation initiatives put in place. The main objective of the consolidation service is to maximize resource exploitation while minimizing energy consumption and costs, among others. Even so, we have noticed that the reported literature doesn't treat license management in the cloud environment as a whole, especially, from the resource management perspective and the overwhelming majority of the consolidation work focuses on resource optimization at the IaaS level. Therefore, we propose a reinforcement learning-based scheme that allows efficient use of resources and optimizes costs, energy consumption, and resource wastage, while remaining compliant. The experimental results show that our intelligent consolidator outperforms the baseline approaches according to the evaluation metrics used regardless of the resource heterogeneity and the data center dimensionality.

Keywords Compliance · Cloud data centers · Energy-efficiency · Artificial intelligence · Resource management · Containerization · Reinforcement Learning

Mathematics Subject Classification 90B50 · 68M14 · 68T99 · 68M20 · 68U01

✉ Leila Helali
leilahelali.ing@gmail.com

Mohamed Nazih Omri
mohamednazih.omri@enisu.u-sousse.tn

¹ MARS Research Laboratory LR17ES05, University of Sousse, Sousse, Tunisia

1 Introduction

With the emergence of new paradigms like softwarization and network virtualization, the demand for services and resources in cloud data centers (DCs) was accelerated. The SaaS market represents the most important evolution that remains in 2021, with \$97 billion in global spending [1]. Proprietary solutions are particularly underlined due to the current economic climate. Managing these services protected by licenses represents a recent challenge where more than 50% of the incurred costs within a DC are reserved to software licensing, maintenance, and deployment [2]. Indeed, the software license charges (software costs) may be included in the VM price (License Charges Included) when VM-based virtualization technology is used or considered separately. In the latter case, the cloud user, when deploying the software, considers his software license (Bring Your Own License) [3]. Here, we consider the second scenario. In addition to the generated costs, non-compliant situations can cause many potential risks which translate into economic losses. This notion of compliance appeals to the metric model used in software licensing which remains obscure in commercial cloud data centers. The most popular licensing method uses declarative licenses, where the business agreements established are periodically audited to check compliance [4]. In highly dynamic cloud environments, when resources and licenses should be scaled up and down rapidly according to the real needs, this method is unsuitable and costly. Moreover, software licensing is considered as an obstacle for the cloud [5]. Here, we consider the processor-based licensing, which depends on the number of cores of the processor and a core processing license factor as in the Oracle database case [6].

From a resource management perspective, energy consumption represents another important component of operational expenditure in the cloud and can contribute to global warming due to the enormous emission of carbon and the released temperature. The amount of energy is significant in the actual big data centers delivering cloud computing services that contain thousands of computational machines [7]. Energy consumption is increased with increasing resource utilization within a DC. The poor management of resources can cause exploitation of only 10% of about 30% cloud servers [8]. As a result, around 3-13% of global electricity will be used by data centers by the year 2030 [9].

To cope with the aforementioned problems, consolidation is a suitable technique [8, 10] that allows to run more workloads with lower costs and lowered power demands without adding new hardware and resources [8]. Through migration and optimal allocation, software licenses are dynamically provisioned and released elastically according to the real use to prevent under/over utilization of resources. Thus, our approach tends to maximize the DC utilization by minimizing the wasted resources at physical machine (PM) and virtual machine (VM) levels and reduce energy consumption along with costs including license and VM costs, while being compliant.

To this end, we propose a compliance-aware reinforcement learning (RL)-based intelligent scheme (COMPaRL) to dynamically allocate the migrated software. It is a proactive approach that handles the compliance and checks the number of installed instances of each software from the resource scalability step, instead of reactively auditing the already installed software instances to deal with the compliance considerations. In our model, we consider the heterogeneity of resources at VM and PM and

the dimensionality of the DC to treat the case of small to relatively big data centers. These DCs are virtualized according to the VM-container architecture when software license (SL) applications are packaged in containers that run in VMs since software licenses require strict security measures. This virtualization solution represents the best alternative until now [8]. This also is encouraged by the quick deployment of new SLs along with the agility and lightweight of containers to develop a dynamic consolidation solution in dynamic and highly changing cloud environments.

The following points summarize the key contributions to address the mentioned challenges:

- (i) An approach for migratable SL candidates selection, the Maximum License Cost (MLC) policy, is implemented. This allowed important savings via migration by saving the cost of the most expensive software licenses.
- (ii) An intelligent software license consolidation (SLC) scheme is proposed which dynamically reallocates the SLs based on the RL framework.
- (iii) A compliance checking algorithm that allows deciding if an allocation is compliant with the processor-based metric before performing the new placement.
- (iv) Extensive simulations are conducted to evaluate our proposed scheme considering different metrics and approaches which show the superiority and the effectiveness of our approach.

The rest of this paper is structured as follows. In Sect. 2, we present the related work followed by the formulation of our proposed model in Sect. 3. Section 4 describes our proposed architecture and its building blocks. In Sect. 5, our intelligent consolidation approach applying reinforcement learning is presented. The simulation settings and results analysis are detailed in Sect. 6. Section 7 discusses the major findings and insights of our work and gives an overview of overheads and scalability through the consolidation process. While Sect. 8 concludes the work and gives some future outlook.

2 Related work

The workload consolidation at the software level while considering licenses has not yet taken his interest in the scientific community. Only Sen et al. [2], in their work, consider software license consolidation, seeking to place workloads with similar software stack on the same PM to reduce the number of software instances, and thus, costs. The authors consider a typical CRM-web application and they consolidate the software into several layers. In this work, a classical hardware consolidation (VMs) and a per server consolidation of software are performed. In the latter, the workload of the same application is grouped into the same software instances within the same server. This approach suffers from isolation problems with the lack of dependencies and support to migrate hosted applications across servers. It also has a security problem because the fact of consolidating in the stack above the VM without taking security measures, if a software (application, portal, middleware) is contaminated, it incurs serious security problems. In our work, the containers that package the software licenses and their dependencies resolve the isolation problem, and the considered VM-container virtualization architecture allows to resolve the security risks [8]. In [11], Mann considers

Table 1 Summary of related works on software and/or license consolidation

Work	Consolidation level	Compliance	License cost	VM cost	Energy	Resource wastage
Mann [11]	VM	-	✓	-	-	-
Sen et al. [2]	Hardware, software	-	✓	-	-	-
Tchana et al. [12]	Software, VM	-	-	✓	✓	-
Our approach COMPaRL	Software license	✓	✓	✓	✓	✓

the cost of licensed components in virtual machine consolidation. The author seeks for the selection of suitable VMs to place software components and suitable PMs to accommodate these VMs, without considering the deployment or consolidation aspects of these components. Whereas, in our work, we consolidate at the software level and not the VM. In [12] a software consolidation approach was proposed to optimize energy consumption and costs that do not consider licenses. Another approach to multi-objective application consolidation was proposed in [13]. It deals with energy consumption and performance costs. In this work, application consolidation was considered as VM + container migration and evaluated with some heuristics. Mainly the operating costs (energy consumption) and the migration costs were discussed. None of this work considers the resource wastage or compliance aspects (see Table 1).

In [14] and [15], the consolidation of software licenses to optimize costs and energy was discussed. In these works, heuristic solutions based on the well-known Best Fit Decreasing (BFD) and First Fit (FF) algorithms [16] which are very popular in VM consolidation works have been proposed. These solutions have shown motivating results but compliance and wastage of resources have not been taken into account.

Some virtual machine consolidation work treated resource wastage like [17] and [18]. In the multi-objective consolidation works like [13, 17], the weighted sum was used to transform the optimization problem into a single objective problem.

Technically speaking, heuristics, metaheuristics, and reinforcement learning-based solutions are the most adopted in resource consolidation. A comparison is given here [8]. We noticed that RL approaches surpass heuristic and metaheuristic solutions [8]. [19] and [20] reported very promising results for virtual machine consolidation based on reinforcement learning.

The work of [19] seeks a trade-off between energy consumption and performance without considering resource wastage, costs, or compliance. The works of [20] and [17] treated only VM placement without migration while in our work, we proposed a whole dynamic consolidation process of software licenses. Moreover, these two works treat only power consumption and resource wastage at the server level. In addition to these objectives, in our work, we treat the resource wastage at the VM level and costs while proposing a compliance checker algorithm that allows only compliant allocations to be performed.

	$SL_1^r.r1$	$SL_1^r.r2$	\dots	$SL_1^r.r_q$
	$SL_2^r.r1$	$SL_2^r.r2$	\dots	$SL_2^r.r_q$
$SL_1^1.r1$	$SL_1^1.r2$	\dots	$SL_1^1.r_q$	$SL_2^2.r_q$
$SL_2^1.r1$	$SL_2^1.r2$	\dots	$SL_2^1.r_q$	$SL_k^2.r_q$
$SL_k^1.r1$	$SL_k^1.r2$	\dots	$SL_k^1.r_q$	

Fig. 1 The set of SLs with their resource utilization

Regarding compliance in the cloud, in [21], the authors treat the compliance in the instantiation step at the time of the software deployment. For that, they proposed a modeling of the metrics and the necessary information about the software product to automatize the compliance management, and they do not consider the migration and dynamic reallocation of software licenses.

Furthermore, in [22] and [23], another aspect of compliance in cloud systems were addressed. The two works proposed data privacy solutions based on Blockchain technology to support the General Data Protection Regulation (GDPR) in Europe. In the work of Aujla et al. [22], a compliance-aware cloud application engineering was proposed to improve transparency on the use of personal data. In the second work, Barati et al. proposed a new approach for monitoring, auditing, and verifying the operations performed on a user's personal data in the cloud.

Besides this important aspect of data compliance, we address here application compliance with the processor-based licensing model. We tackle application compliance throughout the management and scalability of resources at software and hardware levels in cloud systems.

3 System model and problem formulation

The SLC is an NP-hard [12] problem modeled as a bin-packing problem. We consider a set of physical servers $PM = \{PM_i | i = 1, \dots, N_{PM}\}$ and a set of virtual machines $VM = \{VM_j | j = 1, \dots, N_{VM}\}$ with associated resource usage values to execute the SLs. We consider also a set of software $S = \{S_k | k = 1, \dots, N_S\}$ offered by the SaaS provider with q resource utilization (Fig. 1) and the associated licenses and fees. The set of N_l licenses for each software S_k is $L = \{L_r | r = 1, \dots, N_l\}$ (Eq. 4). The set of software licenses is denoted $SL = \{SL_k^r | k = 1, \dots, N_S, r = 1, \dots, N_l\}$.

They are presented in the Fig. 1 with their resource (CPU, memory, bandwidth, etc.) requirements. r_i designates the resource i , $i = 1, 2, \dots, q$.

The relationship that defines if a license belongs to software is formally given by the Eq. 1.

$$\tau_k^r = \begin{cases} 1 & S_k \text{ is licensed under } L_r \\ 0 & \text{else} \end{cases} \quad (1)$$

The relationship between a software license SL_k^r and a virtual machine VM_j , a VM_j and a PM_i are given by the Eqs. 2 and 3 successively.

$$\beta_j^k = \begin{cases} 1 & SL_k^r \text{ is allocated to } VM_j \\ 0 & \text{else} \end{cases} \quad (2)$$

Finally, we express the relationship between a software license SL_k^r and a physical machine PM_i as $\beta_j^k \delta_i^j$ with $i \in [1..N_{PM}]$, $j \in [1..N_{VM}]$ and $k \in [1..N_S]$.

$$\delta_i^j = \begin{cases} 1 & VM_j \text{ is placed on } PM_i \\ 0 & \text{else} \end{cases} \quad (3)$$

Our goal is to optimize the total cost (license+VM) along with the overall energy consumption while minimizing the wasted resources within the DC.

Table 2 summarizes the notations used in the article.

3.1 Processor-based metric and compliance models

Here, we consider a processor-based model use case of Oracle Database. Thus, following this metric, the number of required licenses shall be determined by multiplying the total number of cores of the processor by a core processor factor. All cores on all multicore chips for each licensed program are to be aggregated before multiplying by the appropriate core processor licensing factor and all fractions of a number are to be rounded up to the next whole number" [6]. Hence, for a given PM_i , the number of required licenses N_{Li} for an application licensed under the processor-based model is given by the Eq. 4.

$$N_{Li} = N_{ci} * C_f \quad (4)$$

Where N_{ci} is the total number of cores (see Eq. 5), C_f in the core factor, N_{pi} is the number of processor per server, and C_{pi} is the number of cores per processor. For example, to license a processor with 6 cores and a core factor of 0.25, we need 2 licenses which is the aggregation of 1.5 ($6*0.25$).

$$N_{ci} = N_{pi} * C_{pi} \quad (5)$$

To ensure compliance with the considered license model, we propose the Eq. 6 to account for the number of licenses in each physical machine PM_i for each software

Table 2 Symbols and notations

Symbol	Description
N_{PM}, N_{VM}	Number of PMs and VMs in a DC, respectively
N_S	Number of running software in a DC
N_l^k	Number of licenses for each software S_k
τ_k^r	A binary variable indicating if the SL_k^r belongs to the software S_k and is licensed under L_r license
β_j^k	A variable that indicates if the software license SL_k^r is running in the VM_j
δ_i^j	A variable that indicates if the VM_j is hosted by the PM_i
$N_{L_i}^k$	Number of required licenses of a software S_k for a PM_i
N_{c_i}	Total number of cores of a PM_i
C_f^k	A core licensing factor of a software S_k
N_{p_i}	Number of processors per server PM_i
C_{p_i}	Number of cores per processor in a server PM_i
$L_{PM_i}^k, L_{VM_j}^k$	Number of licenses of the software S_k in a PM_i and VM_j , respectively
L_r	Number of licenses of a given software
NU_i^q, NU_j^q	Normalized utilization of resource q of the PM_i and VM_j , respectively
NU_k^q	Normalized utilization of resource q of the SL_k
Cap_i^q	Capacity of resource q of the PM_i
NU_i^c, NU_j^c	Normalized utilization of CPU resource of the PM_i and VM_j , respectively
NU_k^c	Normalized utilization of CPU resource of the SL_k
Cap_i^c	Capacity of CPU resource of the PM_i
NU_i^m, NU_j^m	Normalized utilization of memory resource of the PM_i and VM_j , respectively
NU_k^m	Normalized utilization of memory resource of the SL_k
Cap_i^m	Capacity of memory resource of the PM_i
P_i^{st}, P_i^{dy}	Static and dynamic power consumption of the PM_i , respectively
P_i^{fu}	Maximum power consumption of a server when is 100% utilised
$NU_i^c(t)$	Normalized CPU utilization of the PM_i at time t
$P_i(NU_i^c(t))$	Power utilization of the PM_i at time t
E_i^s	Energy consumption of the PM_i
E	Total energy consumption in the DC
PE_i	Power efficiency parameter of a PM_i
RW_i, RW_j	Resource wastage of the PM_i and VM_j , respectively
NR_i^t, NR_i^o	Normalized residual resources of the PM_i
NR_i^c, NR_i^m	Normalized residual CPU and memory resources of the PM_i , respectively
NR_j^t, NR_j^o	Normalized residual resources of the VM_j
NR_j^c, NR_j^m	Normalized residual CPU and memory resources of the VM_j , respectively
NR_i^{min}, NR_i^{max}	Minimum and maximum residual resources of CPU and memory of PM_i , respectively

Table 2 continued

Symbol	Description
NR_j^{min}, NR_j^{max}	Minimum and maximum residual resources of CPU and memory of VM_j , respectively
RW	Total resource wastage in the DC
ρ, η	Small positive numbers
Lf_{VM_j}	License fees in a VM_j
$Lf_{SL_k^r}$	License fees of the SL_k^r
Rf_{VM_j}	Renting fees of the VM_j
C_j, C_i	Total cost in a VM_j and PM_i , respectively
C	Total cost in the DC
f, f_i	Objective functions
W_i	Weights ($i = 1, 2, 3$)
T_{mig}	Migration time of an SL
B	Bandwidth

S_k . It is estimated by summing the number of licenses ($L_{VM_j}^k$) in all VM s hosted in PM_i .

$$L_{PM_i}^k = \sum_j \beta_j^k \delta_i^j L_{VM_j}^k = \sum_j \sum_r \tau_k^r \beta_j^k \delta_i^j (L_r) \quad (6)$$

3.2 Resource utilization model

We define the normalized utilization vector of a PM_i as:

$$NU_i^q = \sum_j \frac{\delta_i^j NU_j^q}{Cap_i^q} = \sum_j \sum_k \frac{\beta_j^k \delta_i^j NU_k^q}{Cap_i^q} \quad (7)$$

The normalized CPU utilization (NU_i^c) of a PM_i is estimated by the sum of the utilization of all software running in all VMs hosted in the PM and is given by Eq. 8. NU_j^c and NU_k^c are the normalized CPU utilization of a VM_j and an SL_k successively, and Cap_i^c is the CPU capacity of the PM_i .

$$NU_i^c = \sum_j \frac{\delta_i^j NU_j^c}{Cap_i^c} = \sum_j \sum_k \frac{\beta_j^k \delta_i^j NU_k^c}{Cap_i^c} \quad (8)$$

The normalized memory utilization of a PM_i is given by Eq. 9. Where NU_j^c , NU_k^m and Cap_i^m represent the normalized memory utilization of the VM_j , that of the SL_k^r ,

and the memory capacity of the PM_i successively.

$$NU_i^m = \sum_j \frac{\delta_i^j NU_j^m}{Cap_i^m} = \sum_j \sum_k \frac{\beta_j^k \delta_i^j NU_k^m}{Cap_i^m} \quad (9)$$

3.3 Power and energy models

The power consumption of a machine PM_i is modeled as the sum of the static (P^{st}) and dynamic (P^{dy}) parts as shown in Eq. 10.

$$P_i = P_i^{st} + P_i^{dy} \quad (10)$$

The former is defined by the power consumption of the host when no VM is active and thus no SL runs on the machine. Otherwise, the dynamic part is added to obtain the total power of a server which is detailed in Eq. 11.

The popular power consumption model [8] defines the power consumption of each server as a linear function of the CPU.

$$P_i(NU_i^c(t)) = \begin{cases} P_i^{st} + (P_i^{fu} - P_i^{st}) * NU_i^c(t), & NU_i^c(t) > 0 \\ 0, & \text{otherwise} \end{cases} \quad (11)$$

The energy consumed by a server i is given by the Eq. 12.

$$E_i^s = \int_{t_1}^{t_1 + \Delta t} P_i(NU_i^c(x)) dx \quad (12)$$

This allows estimating the total energy consumed in a DC by Eq. 13.

$$E = \sum_{i=1}^{N_{PM}} E_i^s \quad (13)$$

3.4 Resource wastage model

Inspired by [17], we propose a generalized formula to estimate the resource wastage for q -dimensional PM resources (see Eq. 14).

$$RW_i = \frac{\sum_{t=1}^q |NR_i^t - \min_{o=1}^q (NR_i^o)| + \eta}{\sum_{t=1}^q NU_i^t} \quad (14)$$

As we consider the CPU and memory resources, this equation became (see Eq. 15):

$$RW_i = \frac{|NR_i^{max} - NR_i^{min}| + \eta}{NU_i^c + NU_i^m} \quad (15)$$

NR_i^t is the normalized residual resource (CPU, memory) of the host PM_i and is given by the Eq. 16.

$$NR_i^t = 1 - NU_i^t \quad (16)$$

NR_i^{max} denotes the maximum residual resources of memory and CPU utilization of the given PM_i (see Eq. 17).

$$NR_i^{max} = \max(NR_i^c, NR_i^m) \quad (17)$$

NR_i^{min} denotes the minimum residual resources of memory and CPU utilization of that PM (see Eq. 18).

$$NR_i^{min} = \min(NR_i^c, NR_i^m) \quad (18)$$

In the same way, we propose the Eq. 19 below to model the resource wastage of a q-dimensional VM_j .

$$RW_j = \frac{\sum_{t=1}^q |NR_j^t - \min_{o=1}^q (NR_j^o)| + \rho}{\sum_{t=1}^q NU_j^t} \quad (19)$$

NR_j^t is the normalized residual resource of the virtual machine VM_j and is given by the Eq. 20.

$$NR_j^t = 1 - NU_j^t \quad (20)$$

NR_j^{max} and NR_j^{min} denote the maximum (respectively minimum) residual resources of memory and CPU utilization (see Eqs. 21 and 22) of a given VM_j .

$$NR_j^{max} = \max(NR_j^c, NR_j^m) \quad (21)$$

$$NR_j^{min} = \min(NR_j^c, NR_j^m) \quad (22)$$

Thus, the Eq. 19 became:

$$RW_j = \frac{|NR_j^{max} - NR_j^{min}| + \rho}{NU_j^c + NU_j^m} \quad (23)$$

ρ and η are small positive numbers. We assume that $\rho = \eta = 0.0001$ [17]. The total resource wastage in the DC is given by Eq. 24.

$$RW = \sum_{i=1}^{N_{PM}} (RW_i + \sum_j \delta_i^j RW_j) \quad (24)$$

3.5 Cost model

The cost of our model is formed by the fees paid for renting a VM (Rf_{VM_j}) within an hour along with the license cost (Lf_{VM_j}) which is estimated by the sum of the license

fees ($Lf_{SL_k^r}$) of the software running in a given VM (Eq. 25).

$$Lf_{VM_j} = \sum_k \tau_k^r \beta_j^k Lf_{SL_k^r} \quad (25)$$

As we consider processor-based licensing, the license fee is paid once for at least an instance of software running in a given machine. The total cost of a VM is given by Eq. 26.

$$C_j = Rf_{VM_j} + Lf_{VM_j} \quad (26)$$

In the same way, the cost of a PM_i is estimated by the sum of the cost of all VMs running on it (Eq. 27).

$$C_i = \sum_j \delta_i^j C_j \quad (27)$$

Finally, the total cost within a DC is given by the Eq. 28.

$$C = \sum_{i=1}^{N_{PM}} C_i \quad (28)$$

3.6 Problem formulation

The objective functions that reflect the minimization problems are given by Eq. 29.

$$f = \begin{cases} \min(E) = \sum_{i=1}^{N_{PM}} E_i \\ \min(C) = \sum_{i=1}^{N_{PM}} C_i \\ \min(RW) = \sum_{i=1}^{N_{PM}} (RW_i + \sum_j \delta_i^j RW_j) \end{cases} \quad (29)$$

To transfer our optimization problem into a mono-objective problem that combines the energy, resource wastage, and cost objective functions in one function f , we adopt the most commonly used linear weighted sum aggregation (Eq. 30) [24]. The weights w_i are comprised in the interval $[0,1]$ and verify the Equation $\sum_{i=1}^n w_i = 1$.

$$f = \sum_{i=1}^n w_i * f_i \quad (30)$$

In our case, $n=3$ and f became :

$$\min(w_1 * E + w_2 * C + w_3 * RW) \quad (31)$$

w_1 , w_2 and w_3 reflect the importance of each objective function in the single-objective function f . Thus, the final result of the proposed model depends on these three variables. Inspired by [13] and [17], we consider the three objectives of a comparable magnitude and consider equivalent weights. It is worth noting that normalized values of the three objectives are considered in the weighted sum function to homogenize the

different objectives since they have different units. Thus, power consumption within a PM is normalized by the initial consumption of the DC, and the total cost of a VM is normalized by the total cost within a PM to have values comparable to resource wastage. Furthermore, the following constraints are considered:

$$L_{PM_i}^k \leq N_{l_i} \forall i \in [1..N_{PM}], k \in [1..N_{NS}] \quad (32)$$

The Eq. 32 denotes that the allowed software instances of a given license type must not exceed the allowed number of that license type following the processor-based metric.

$$\sum_j \delta_i^j NU_j^c < Cap_i^c \forall i \in [1..N_{PM}], j \in [1..N_{VM}] \quad (33)$$

Equations 33 and 34 fix the constraints related to the memory and CPU utilization limit of a PM which should not be exceeded by CPU and memory utilization of all hosted VMs.

$$\sum_j \delta_i^j NU_j^m < Cap_i^m \forall i \in [1..N_{PM}], j \in [1..N_{VM}] \quad (34)$$

Likewise, Eqs. 35 and 36 allow ensuring that the CPU and memory utilization of software licenses that are co-located in a VM does not exceed the limit of CPU and memory utilization of the VM hosting them.

$$\sum_k \beta_j^k NU_k^c < Cap_j^c \forall j \in [1..N_{VM}], k \in [1..N_{SL}] \quad (35)$$

Cap_j^c and Cap_j^m represent the resource capacity in terms of memory and CPU of the VM_j , successively.

$$\sum_k \beta_j^k NU_k^m < Cap_j^m \forall j \in [1..N_{VM}], k \in [1..N_{SL}] \quad (36)$$

Equations 37 and 38 ensure that each software license is exactly mapped to one VM and each VM is hosted by one PM at a time, successively.

$$\sum_{j=1}^{N_{VM}} \beta_j^k = 1 \forall k \in [1..N_{SL}] \quad (37)$$

$$\sum_{i=1}^{N_{PM}} \delta_i^j = 1 \forall j \in [1..N_{VM}] \quad (38)$$

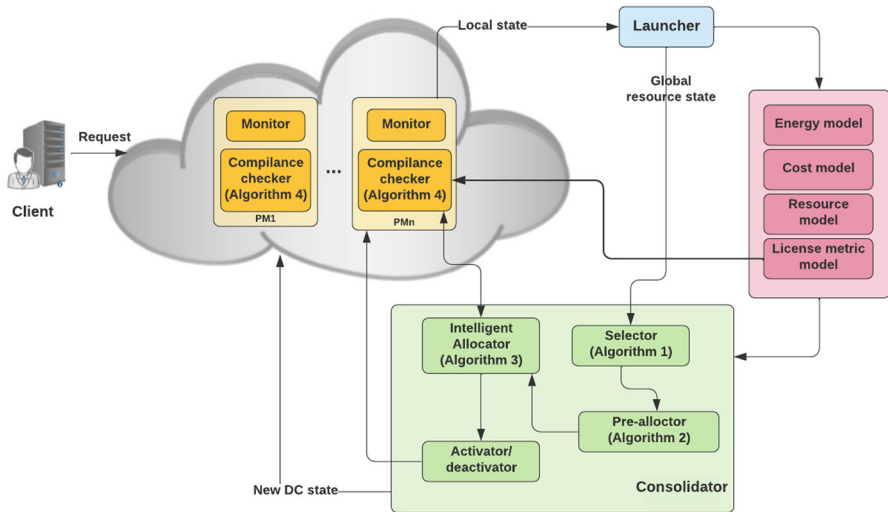


Fig. 2 System architecture of COMPaRL

4 COMPaRL architecture

Here, we detail the proposed architecture for COMPaRL given by Fig. 2 and their building blocks.

In our proposition, we separate the monitoring and compliance checking which are distributed parts deployed in every compute host from the allocation of software licenses instead of a fully centralized architecture as in Entropy [25]. Our proposed distributed architecture regarding resource monitoring and compliance checking allows the natural scalability of the system and avoids the single point of failure. The monitor is responsible for gathering information about resource utilization and getting local resource states which will be used by the launcher to trigger the consolidation process. Based on this information, the modeler estimates the cost, energy, and resource wastage in the DC, and the selector chooses the software license to migrate according to an optimal strategy named MLC that will be detailed in the next section. Then, the intelligent allocator selects, in a first step, the list of virtual machine candidates for each migrated SL and evaluates the potential allocation to, finally, decide of the most suitable actions that optimize the DC state based on the RL framework. The allocator performs a preselection step as will be discussed below. In the next step, the allocation decision is communicated to the compliance checker to proactively verify if a given allocation doesn't escape to compliance by verifying if the number of licensed instances of the same software surpasses the allowed amount based on the processor-based metric. If this is the case, the compliance module triggers the alert (see Algorithm 4) and prevents the allocator to evaluate another allocation and the process is repeated until a compliant allocation will be found. The last decision of SL placement is then communicated to the activator/deactivator module to activate new VMs/PMs when needed and/or stop unused ones. Finally, the DC state is updated to dynamically start a new consolidation.

5 Compliance-aware RL-based SLC

The software license consolidation follows a whole process illustrating the main steps of the proposed intelligent scheme to perform the DC optimization in terms of energy consumption, resource wastage, and total costs. These steps, detailed below, are performed at runtime according to the utilization level of the PMs. The DC state is periodically evaluated and the consolidation decision is made based on resource state.

5.1 Consolidation decision

The first step concerns the identification of critical machines. This answers the question of when to perform consolidation and migration. In our case, the triggering point that initiates the consolidation process is based on the physical machine status detection (over/underloaded). The status of a host is considered as overloaded (OL) if its utilization reaches or exceeds a static T_{ol} threshold. As well, the underloaded (UL) status is controlled by a static T_{ul} threshold. If a machine is overloaded, some software licenses will be selected for migration to resolve the overloaded state. For underloaded hosts, all running software licenses will be migrated and the PMs switch to the sleep mode after the migration ends. The process is iteratively and dynamically repeated for all PMs.

5.2 Migration

After triggering the consolidation process, some SLs have to be selected for migration. For this step, we propose a novel strategy named MLC that saves the cost of the most expensive SLs candidates of migration.

Algorithm 1 Maximum License Cost for Migration

Require: overloadPM
1: $mx \leftarrow 0$
2: **for** $VM \in \text{overloadedPM}$ **do**
3: $maxC \leftarrow mx$
4: $selectedSL \leftarrow NULL$
5: **for** $SL \in VM$ **do**
6: $C \leftarrow SL.getCost()$
7: **if** $C > maxC$ **then**
8: $maxC \leftarrow C$
9: $selectedSL \leftarrow SL$
10: **end if**
11: **end for**
12: **end for**
13: **return** selectedSL

On a critical machine, the critical SL(s) that have the maximum license fees will be given the highest priority to be migrated and it is added to the list of migratable SLs. The idea is that if we can migrate the software license with the maximum cost

in a VM which contains another instance of the same software, we save the cost of the migrated license. This gives more chance to optimize costs at the maximum and is given by the Algorithm 1.

5.3 Allocation: destination preselection

The main idea behind our solution is to select the list of candidate VMs which contains at least one instance of the software in question belonging to the same license. In this case, no license fee would be incurred with this mapping which makes those VMs very promising for the current allocation (Algorithm 2).

Algorithm 2 Pre-selection of Promising VMs

Require: $SLToMigrateList$, PM_List

```

1:  $SLToMigrateList.sortDecreasingUtilization()$ 
2: for  $SL \in SLToMigrateList$  do
3:    $L \leftarrow getLicenseType(SL)$ 
4:   for  $PM \in PM\_List$  do
5:     for  $VM \in PM$  do
6:       if there is a software instance in  $VM$  with license  $L$  then
7:         add  $VM$  to  $L\_VM$ 
8:       end if
9:     end for
10:   end for
11:    $select\_VMs(randomVMs)$ 
12:    $sortDecreasingOrder(randomVMs)$ 
13:   add  $randomVMs$  to  $L\_VM$ 
14: end for
15: return  $L\_VM$ 

```

Firstly, the list of migratable SLs is sorted (line 2) according to the CPU utilization. Then, for each SL, the type of the license for a given software is determined (line 3) to check all the VMs within the DC that contains at least one SL of the same license and add them to the list of promising VM destinations. This list is augmented by some random VMs sorted in decreasing order to extend the search space and promote the fullest ones to maximize resource utilization (line 5-11).

5.4 Multi-objective RL model for dynamic SL placement

An important step when applying reinforcement learning to a practical problem is the identification of the state space, action space, and reinforcement signal (reward function).

5.4.1 State space

The state space which is denoted by $S_t = \{S_{t_1}, S_{t_2}, \dots, S_{t_{N_{PM}}}, SL_t^{cpu}, SL_t^{mem}, SL_t^{VMid}\}$, is formed by the detailed informations of the set of N_{PM} elements at time step t . Each

element of the state space is given by $S_{t_i} = \{S_{t_i}^{status}, S_{t_i}^{tot_{cpu}}, S_{t_i}^{tot_{mem}}, S_{t_i}^{used_{cpu}}, S_{t_i}^{used_{mem}}\}$. $S_{t_i}^{status} \in \{active, inactive\}$ is the status of the server PM_i at the instant t . Allocating an SL_k to a VM_j changes the state of this VM and thus the PM hosting it.

5.4.2 Action space

For each migrated SL, the action space is defined by a binary variable (PLACE/NOT) indicating if the SL will be placed into a VM_j for all the preselected VMs in the previous step if that VM has enough resources to accommodate the SL. We consider all the VMs in the DC to define the action space and, for a given SL, we evaluate only the allocation to VMs from the preselected list. If a VM_j is not hosted in a PM_i , we consider the combination of the state S_{t_i} and action a_j as impossible.

5.4.3 Reward function

In each step, after executing an action a and performing a new allocation, the agent receives a reward granted by the environment at time t , which is given by:

$$R = \frac{1}{[\frac{1}{3}(E + C + RW)]} \quad (39)$$

The reward is expressed depending on the objective function tending to minimize the energy consumption, costs, and resource wastage.

5.4.4 Q-learning based proposed algorithm

Our RL algorithm (see Algorithm 3) is based on the Q-learning technique. Without any prior knowledge, in the Q-learning algorithm which provided important results in VM consolidation work like [19, 26], the agent computes and stores a Q-values $Q(s,a)$ according to the Eq. 40 in a matrix called the Q-table for each state s and action a in each step. γ and α are positive values less than 1 that, successively, represent the discount factor determining the importance of the predicted future reward and the learning rate.

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_a Q(s', a)) \quad (40)$$

The Q-table is formed by the state space in line and the action space in the column. For each PM in the state space, all the VMs in the DC are represented in a column and only the cases that reflect a potential allocation are evaluated. To explain more our strategy, the cases when a given VM is hosted by a PM are considered for potential allocation and the other cases in the Q-table are frozen. Also, only the pre-selected VMs are considered and for the others, the given Q-value doesn't change. This allows important potential savings when considering only the VMs that run a software instance with the same license as the migrated software. Moreover, in this article, we use the ϵ -greedy policy to ensure the trade-off between the exploration

Algorithm 3 Intelligent Consolidator

Require: The state set: $S = s_{ti}, i = 0, 1, \dots, N_{PM}$

- 1: The action set: $A = a_j, j = 0, 1, \dots, N_{VM}$
- 2: The reward, $R = f(S_i, a_j)$
- 3: The action selection policy, $Q(\epsilon - greedy)$
- 4: *initQtable()* ▷ initialize the Q-table
- 5: **for** $ep \in episodes$ **do**
- 6: *generateStates()* ▷ initialize the environment
- 7: **while** *not_final_state* **do**
- 8: $s_t \leftarrow newState()$
- 9: $SList \leftarrow getSelectedSLs()$ ▷ the result of the monitoring step to select the SL to migrate from UL/OL PMs
- 10: **for** $SL \in SList$ **do**
- 11: $VMList \leftarrow getVMCandidates()$ ▷ the list of preselected VMs
- 12: Choose a suitable action a_t from A for the state s_t ▷ the action that verify the resources and compliance constraints
- 13: Perform the action a_t
- 14: Observe the resulted reward r_t and the next state s_{t+1}
- 15: Update the Q_table
- 16: $Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma max_a Q(s_{t+1}, a))$
- 17: $s_t \leftarrow s_{t+1}$
- 18: **end for**
- 19: update the learning rate
- 20: update the discount factor
- 21: **end while**
- 22: **end for**
- 23: **return** $Q - table$

when the agent tries to explore new allocation possibilities and exploitation when it uses the knowledge already acquired.

5.5 Compliance checking

In situations of non-compliance where the consolidator tends to perform the allocation of an SL to a machine that consumed all the licenses of this migrated software, an alert process is triggered according to Algorithm 4. Once receiving this alert, the consolidation agent chooses another allocation that preserves compliance. This checking process allows to claim the non-conformity behavior to choose another compliant allocation, but also, it allows to account for the number of non-conform placements which helps to estimate the potential losses.

6 Model evaluation

In order to reach an optimal data center state in terms of energy consumption, overall cost, and resource utilization, we implemented the proposed scheme and evaluated its performances as will be detailed below. Firstly, we present the research questions we tend to answer through the extensive simulation, then we detail the setting of the parameters before presenting the results and their analysis.

Algorithm 4 The Compliance Checker

Require: *MigratedSL* (to be placed), *destinationVM*

```

1: compliant  $\leftarrow$  false ▷ A boolean variable to control if the situation is compliant or not
2: PMpm  $\leftarrow$  destinationVM.getSourcePM()
3: LicenseI  $\leftarrow$  MigratedSL.getLicenseType()
4: maxLicenses  $\leftarrow$  pm.getNL(I) ▷ return the number of allowed licenses of type I
5: nSLs  $\leftarrow$  pm.getLicenseCount(I) ▷ return the number of licenses of type I in that pm
6: if nSLs + 1  $\leq$  maxLicense then
7:   compliant  $\leftarrow$  true
8: end if
9: return compliant

```

6.1 Research questions

The main research questions addressed in our work are:

- How does the proposed intelligent approach perform compared to the baseline heuristics?
- To what extent can our MLC policy improve costs within a DC compared to the baseline policies?
- To what extent can our approaches improve the energy consumption and resource utilization in homogeneous and heterogeneous DCs regardless of the resources scales?

6.2 Evaluation environment

6.2.1 Simulation setup

The proposed solution was implemented in java language using Eclipse IDE and JDK 8. and the simulations are carried out on Intel® Core™ i3-8100 CPU @ 3.60 GHz, 8 GB RAM, Windows 10 (Professional).

The model of our simulation program is shown in the class diagram of Fig. 3. As stated above, the intelligent RL consolidator agent interacts with the environment representing the data center composed of a set of servers. It manages and updates the different states composing the environment. In addition to the intelligent consolidator, our simulator allows us to perform heuristic-based allocations. We implemented three heuristic solutions, which will be presented in the compared algorithms section.

In our experiments, we consider homogeneous and heterogeneous resource configurations as shown in Table 3. In the homogeneous scenario, all the servers are of type Haswell (Xeon 2695). To take the values of power consumption of different types of hosts used in our experiments at different utilization levels we used the SPECpower benchmark¹. For each scenario, to meet the real need in computing resources, we consider the DC dimensionality and planned to experiment with our proposed schemes in small and relatively large scale resources. For the small scale, we consider a DC with 30 PMs, 100 VMs, 300 SLs (30 software and 10 licenses). For the wide scenario, we consider 100 PMs, 300 VMs, and 1000 SLs (100 software and 10 licenses). Moreover,

¹ <https://www.spec.org/>.

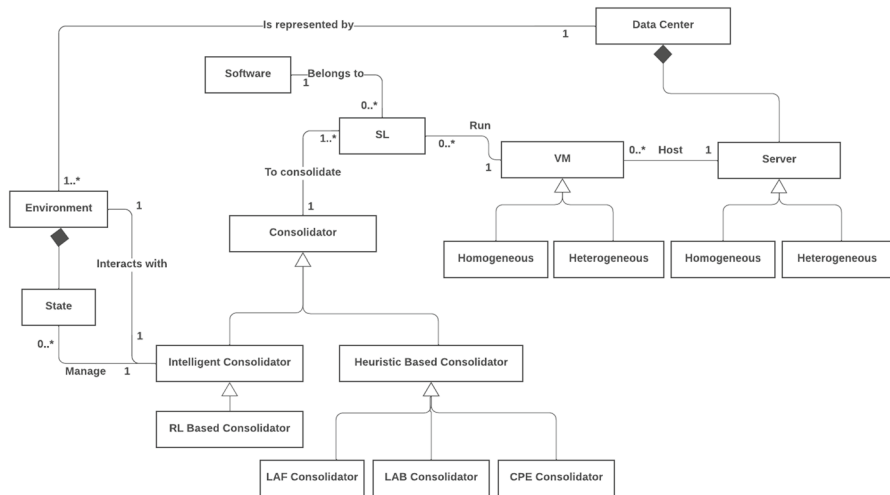


Fig. 3 Class design diagram for simulation modeling of software license consolidation

Table 3 Power consumption model for the used physical hosts

Server type	P^{st} (Wh)	P^{fu} (Wh)	Number of cores	Number of CPU
Haswell (Xeon 2695)	70	120	14	2
Ivy Bridge (Xeon 2670)	65	115	10	2
Sandy Bridge (Xeon 2670)	55	105	8	2

the overload and underload threshold values are 0.8 and 0.4 respectively. In addition, for each PM, we assume a network bandwidth of 1 GB/s and a core factor of 1. For VMs pricing, we use the common Pay as you go model. M1 medium instances VM types are used in the homogeneous scenario, which are charged at \$0.120 per VM per hour. For heterogeneous scenarios, we used Azure VMs² offering the flexibility to virtualize a wide range of solutions supporting windows server, oracle, SAP, etc. The used VMs along with their prices are presented in Table 4 below. The DS11-1 v2 VM types are optimized for DB workloads like SQL Server and are adapted to optimize software licensing costs for customer workloads. The G1 VM type which belongs to G-series supports large DB workloads, specifically SAP HANA, Hadoop, SQL Server, etc. Finally, E8a v4 VMs are ideal for enterprise applications that are memory-intensive. For software pricing, inspired by App Service pricing³ of Azure, we generated the software license fees randomly in [0, 1). For workloads, inspired by [27] and [28] we generated synthetic instances to evaluate our solution. Through normalization, we set the total CPU and memory utilization of a PM as 1 and total CPU and memory utilization of a VM as 0.25. This allows to run up to 4 VMs per host.

² <https://azure.microsoft.com/en-us/pricing/details/virtual-machines/windows/>.

³ <https://azure.microsoft.com/en-us/pricing/details/app-service/windows/>.

Table 4 Virtual Machines pricing

Instance type	Price (\$/hour)
DS11-1 v2	0.149
G1	0.49
E8a v4	0.504

To generate our software synthetic instances, we used the random workload generator given by Algorithm 5 and we set $U_{CPU}^* = 0.025$ and $U_{mem}^* = 0.025$. This allows to run up to 10 SLs in each VM. Then, for each software, a number of licenses are created (initially 10) and their costs are also randomly generated as mentioned above. To simplify concerns, we suppose that all software running in the Dc are licensed). We generated 500 software and created variable licensed instances.

Algorithm 5 Software Synthetic Instances Generation

Require: N_S

```

1: for  $i=1$  to  $N_S$  do
2:    $U_{CPU}^i \leftarrow \text{rand}(2 * U_{CPU}^*)$ 
3:    $U_{mem}^i \leftarrow \text{rand}(U_{mem}^*)$ 
4:    $r \leftarrow \text{rand}(1.0)$ 
5:   if  $(r < P \wedge U_{CPU}^i \geq U_{CPU}^*) \vee (r \geq P \wedge U_{CPU}^i < U_{CPU}^*)$  then
6:      $U_{mem}^i \leftarrow U_{mem}^i + U_{mem}^*$ 
7:   end if
8: end for
9: return  $\langle U_{CPU}^i, U_{mem}^i \rangle \text{ Set}$ 

```

Through preliminary experiments on the correlation coefficient P (see Algorithm 5) varying it in $[0, 0.25, 0.5, 0.75, 1]$ we chose $P = 1$ which showed more stable results for the evaluation scenarios designed beforehand.

Finally, our intelligent RL agent was trained using a learning rate of 0.3, a discount factor, and the exploration/exploitation factor of 1 that are decreased by 99% during the learning process.

6.2.2 Compared algorithms

For the sake of comparison, regarding the SL selection for migration, we compared our MLC policy by:

- Maximum Usage (MU) [29]: this strategy tends to select the software license that has the maximum usage of CPU.
- Minimum Migration Time (MMT) [29]: this is also an adapted version of the MMT policy to the software license consolidation. This policy tends to select the SL that have the minimum migration time, the ratio between the amount of memory of the $SL_k (NU_k^m)$ and the bandwidth B (see Eq. 41).

$$T_{mig} = \frac{NU_k^m}{B} \quad (41)$$

To compare the performance of our intelligent dynamic placement approach, we extended and implemented the following baseline heuristic algorithms proposed here [15].

- License Adapted Best Fit Decreasing (LAB): it is an adapted version of the well-known Best Fit Decreasing (BFD) algorithm [30] to deal with the commercial software consolidation case. This later applies the best fit rules that tend to place the SL in the fullest filled machine that fits it after sorting the migratable SL list in decreasing order of utilization.
- License Adapted First Fit (LAF): it is based on the First Fit algorithm [30] adapted to SL consolidation, which is a rapid placement algorithm, processing the software licenses in a queue and places each SL in the first partially filled VM that can meet its requirements in terms of resources.
- Cost Power-Efficient (CPE): it is an improved version of the LAB using the Eq. 42 to choose the most power-efficient destination machine if two destinations have the same objective values.

$$PE_i = U_i / P_i^{fu} \quad (42)$$

6.2.3 Evaluation metrics

The evaluation metrics that we considered in this work are energy consumption, cost, total resource wastage, number of active PMs, and number of running VMs. They are described as follows:

- Energy consumption: It depends on the number of active machines and the related utilization of resources along with their power consumption.
- Cost: Used to estimate, not only the total cost but also the VM and license cost and show the effects of our MLC policy on these cost components.
- Total resource wastage: This estimates the wasted resources to show the behavior of our proposed scheme on the resource utilization maximization.
- Number of active PMs: It shows the importance of our consolidation scheme on the DC optimization by reducing the number of used PMs.
- Number of running VMs: It can impact the operational costs, especially, the total VM costs which depend on the number of running VMs.

6.3 Results and analysis

In the following, we present the outcomes of our scheme and the analysis of the results to verify the assumed claims above.

6.3.1 Energy consumption results

Concerning the energy consumption experimental results (Fig. 4), in the majority of cases, the MU policy, while combined with the LAF and CPE heuristics and RL approach gave the best results in terms of total energy consumption.

The intelligent approach gave the best savings that reach 68.17% in the large-scale heterogeneous scenario. This is due to the principle that this policy migrates

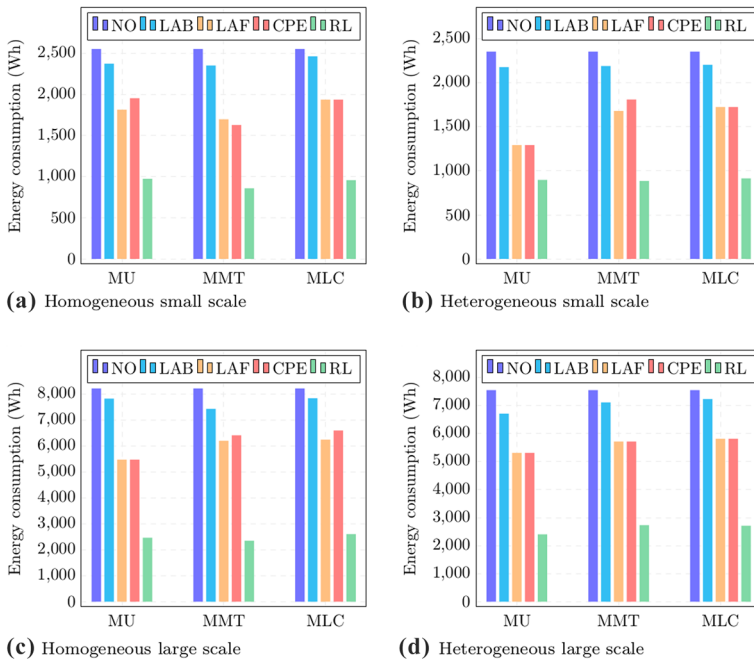


Fig. 4 Energy consumption results in homogeneous/heterogeneous small/large DC

the SLs with the maximum utilization and that allows more chance to maximize the utilization of active VMs. Especially in the heterogeneous scenario, if the target PMs have minimum power consumption values this leads to a minimized energy consumption. Furthermore, we noticed that our intelligent RL approach gave the best results in terms of energy consumption than the heuristics in all the DC dimensions and resource scenarios. Their gain varies between 62.4% in small-scale heterogeneous scenarios and 71.38% in the large-scale homogeneous scenario when combined with MMT policy (Fig. 4).

For the heuristic solutions, the LAB approach gave the worst results since it selects the machines which have the lower resource capacity of CPU, which limits the consolidation possibilities and leads to the use of the maximum number of PMs as can be seen in Fig. 6. The best savings realized by this approach are 11.09% in a large-scale heterogeneous scenario. The CPE approach which is based on LAB surpasses this later since it selects the most power-efficient PM destination. It realized 45.1% gains in the small-scale heterogeneous scenario and uses less number of hosts (see Fig. 6). Except in homogeneous small scale, while combined with the MU policy, the LAF heuristic gave similar results or slightly surpasses the CPE approach in terms of energy. This is because the CPE approach selects the most power-efficient PMs only from the machines with the lower capacity of CPU resources while in the LAF approach, the first available host that can accommodate the SL in question is selected, and not necessarily the fullest one. Even if this led to executing more migrations to

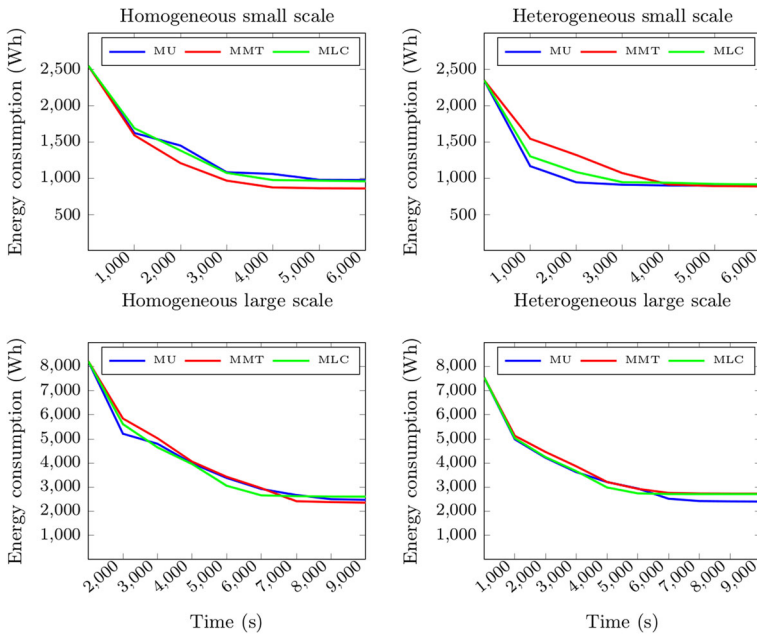


Fig. 5 RL training results of energy consumption in homogeneous/heterogeneous small/large DC

reach an optimal state, this heuristic realized important savings that reached 45.1% in the small-scale heterogeneous scenario.

As we can see in Fig. 5 in the RL training results, in the small scale DCs, the RL agent converges rapidly to the optimized results (at around 2000 episodes), while in large scenarios, it takes more time (around 5000 episodes). This is due to the size of the DC when the wide scenario needs more time to reach an optimal state. In the homogeneous scenario, the MMT policy gave the best energy values as it selects the SLs with minimum memory utilization, which is correlated to the CPU utilization. Thus, while combined with the RL approach, using the ϵ -greedy policy that selects the best action with the maximum reward when the exploitation rate is important, gives the best results. This can be seen in the training results of Fig. 5, especially, in the last episodes when the exploration rate decreases and the agent exploit the acquired competencies.

The results in terms of the used number of PMs are equivalent to the energy consumption results as Fig. 6 shows since the energy consumption is proportional to the CPU utilization and the amount of power consumed by these PMs.

6.3.2 Cost results

Regarding costs, it is worth noting that the total cost results are similar to the license cost which represent the most important part of the total cost. Furthermore, the MLC policy represents the best results in terms of license cost and total cost while combined with all the allocation policies in all scenarios as shown in Figs. 7 and 8. This is due

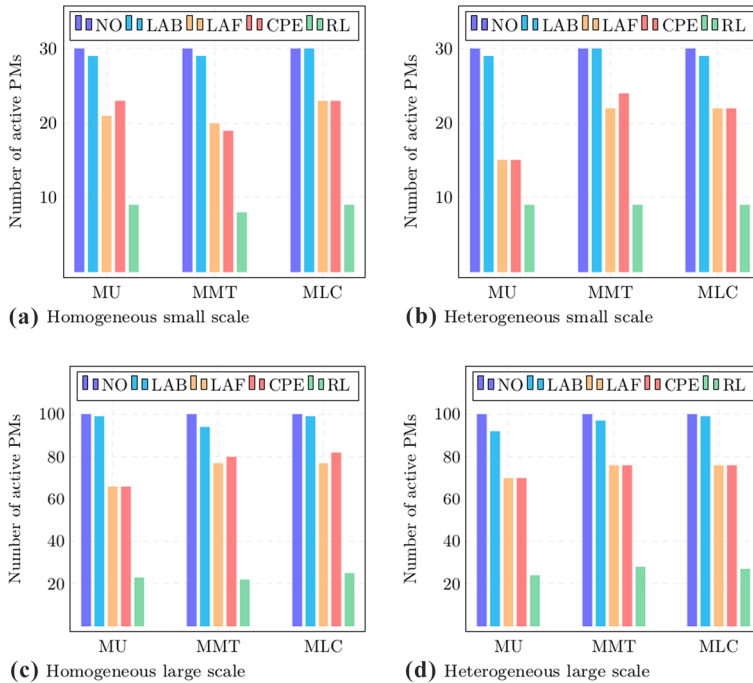


Fig. 6 Number of active PMs in homogeneous/heterogeneous small/large DC

to the act that it migrates the most expensive software license which, once finding a destination machine that accommodates at least one other instance of software with the same license, is paid once according to the processor-based licensing model adopted. This allows saving the cost of the most expensive SLs independently of the placement strategy. The MLC policy, while combined with our RL approach, allowed the best savings that vary from 62.58% savings in small scale homogeneous scenario to 73.49% savings in the wide scenario with homogeneous resources in terms of license costs, and from 61.64% savings to 71% savings in terms of total cost in small scale heterogeneous and large scale homogeneous scenarios successively. The LAB heuristic showed the worst results in all cases because it uses the maximum amount of PMs and also VMs (Figs. 6 and 11) which maximizes the total cost due to the inherent VM cost. Their best gains are estimated by 49.81% savings in terms of license costs and 45.49% in terms of total costs in the large-scale homogeneous scenario. LAF and CPE heuristics showed similar behaviors in the majority of the scenarios when the savings reached 57.71% and 53.9% in small-scale heterogeneous scenario scenarios in terms of license costs and total costs successively. Figure 9 confirms the findings below and shows that the MLC policy gave the best results during the training of the RL agent with remarkable savings compared to the MMT and MU policies. These gains increase with the agent lifecycle. The total VM costs are proportional to the number of running VMs as shown in Figs. 10 and 11 and the gains reached 68.66% in the large-scale homogeneous scenario with 68.66% savings in the number of running VMs by RL_MMT scheme.

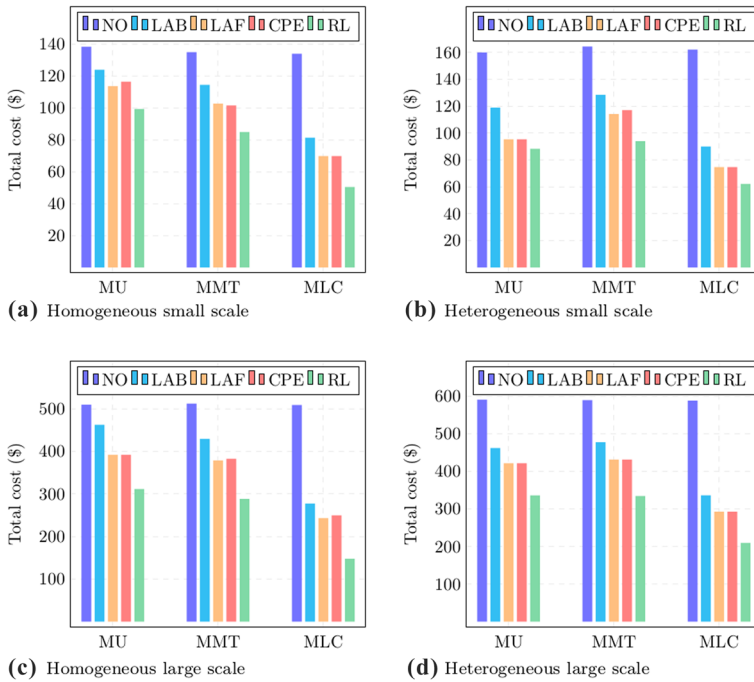


Fig. 7 Total cost in homogeneous/heterogeneous small/large DC

It is worth highlighting that the results in terms of total VM costs and the number of used VMs are identical. As the number of VMs decreases, the user's monetary costs and thus the total cost decrease. This gives more opportunities to reduce the number of active servers and implicitly reduces resource utilization and energy consumption, which guarantees a win-win strategy for both customer and provider in the competitive multi-cloud environments of today. In addition, our approach enables cost minimization not only in heterogeneous scenarios but also in homogeneous DC. This is because the total cost mainly depends on the license cost, which varies depending on the instances of the software that this license belongs to and the license model used.

6.3.3 Resource wastage results

From the Fig. 12 that shows the training results of the RL agent, we notice that with some fluctuations at the beginning, the MU policy converges to optimized results of resource wastage, especially with heterogeneous resources. This is due to the fact that migrating the SLs with the highest CPU utilization allows better utilization of the destination machines and with the heterogeneous scenario, some target machines could accommodate more SLs which maximizes their utilization and thus minimizes the wasted resources.

In addition, if the minimum number of machines are used in a more efficient way, the amount of resource wastage will be minimized which is the case of the RL approach. The heuristic algorithms and RL approach showed similar behavior in terms of wasted

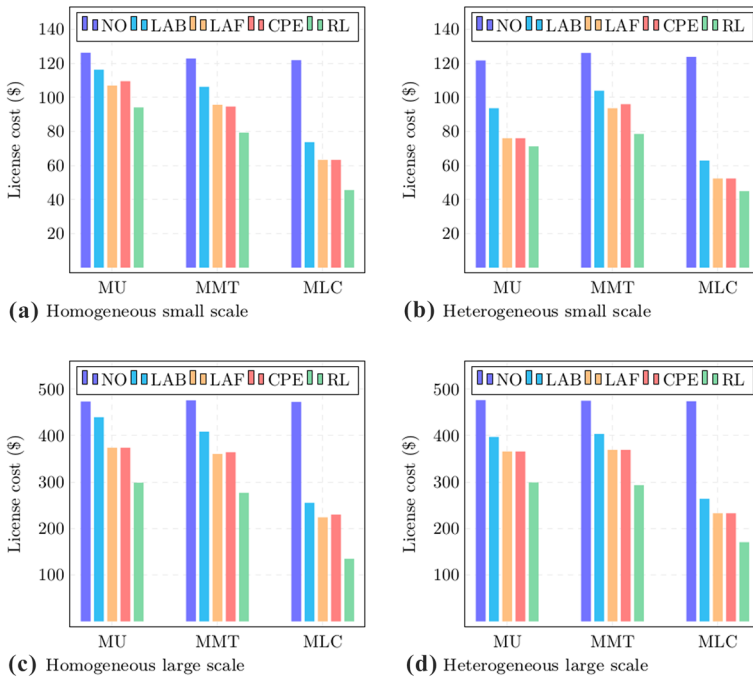


Fig. 8 Licence cost in homogeneous/heterogeneous small/large DC

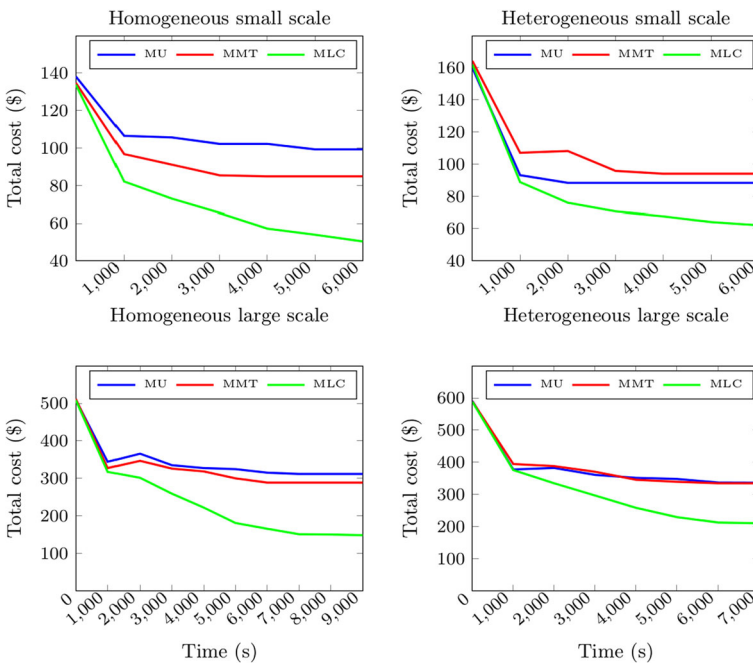


Fig. 9 RL training results of total cost in homogeneous/heterogeneous small/large DC

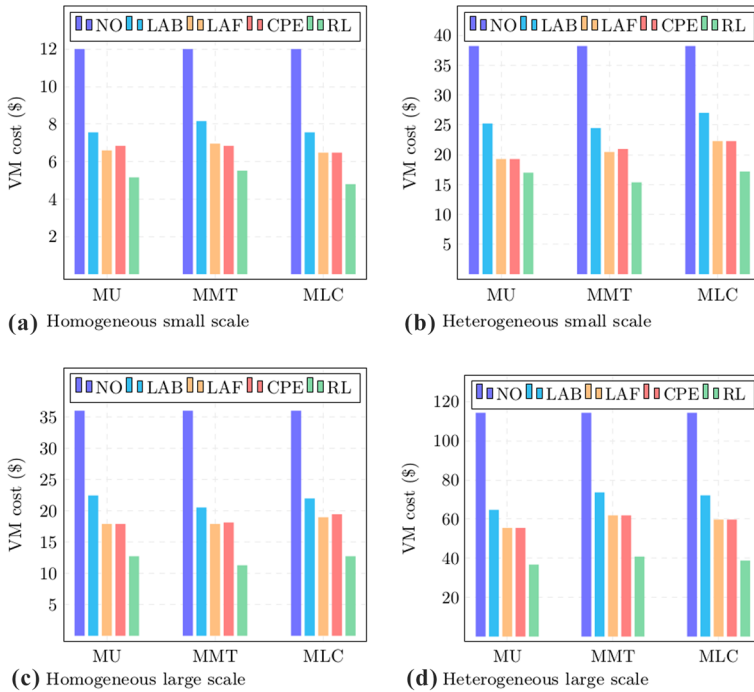


Fig. 10 VM cost in homogeneous/heterogeneous small/large DC

resources. The LAB approach always gave slightly high values because it allows using the maximum number of machines that are less utilized causing a less efficient use of the resource. The best savings are noted in large-scale scenarios that prove the scalability of our approaches and the efficient use of resources that increase with the size of the DC scenarios. The gains reached 77.45% in the wide homogeneous DC.

6.3.4 Sensitivity analysis of UL/OL on DCs

To study the impact of the static thresholds used in the consolidation decision on the energy consumption, costs, and resource wastage, we vary the values of UL and OL thresholds and analyze their effects in this section. We consider UL thresholds of 40%, 50%, and 60%, and OL thresholds of 80%, 90%, and 100%.

Through the results shown in Figs. 13 and 14, we can notice that with all the threshold values, our approach realizes important gains in all the scales with all the resource scenarios. In the majority of cases, a UL equal to 40% and an OL equal to 80% gave the best gains, which validates our choice of the threshold values. Furthermore, despite some fluctuations due to the random workload nature, in more than half of the cases, a UL of 40% gives the best results than 50%, and a UL of 50% gives the best results than 60% while varying the UL threshold (see Fig. 13) with the different optimization objectives. Also, in more than half of the cases, an OL of 80% gives the best results than 90%, and an OL of 90% gives the best results than 100% while

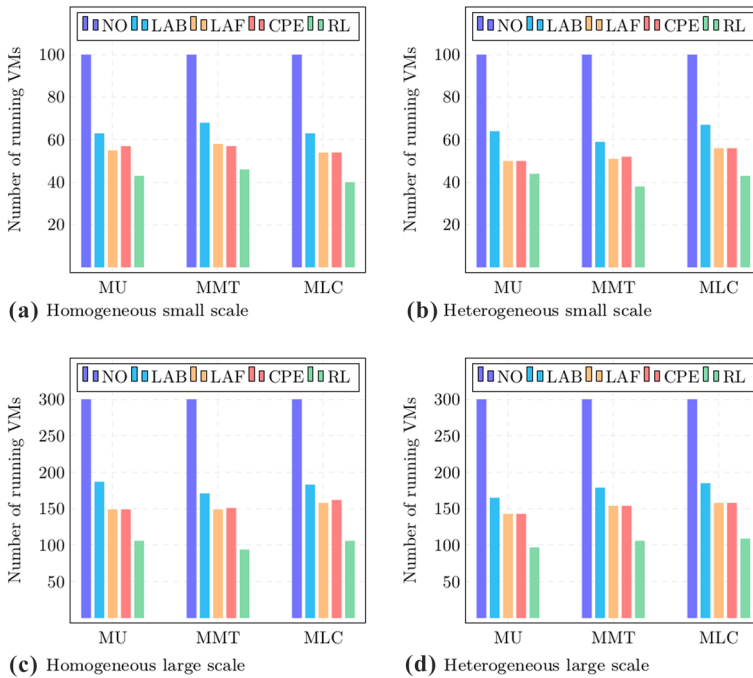


Fig. 11 Number of running VMs in homogeneous/heterogeneous small/large DC

varying the OL threshold (see Fig. 13) with the different optimization objectives. The higher values of UL can cause an over-consolidation when more underutilized servers could be identified. In addition, the higher values of the OL threshold can increase the likelihood that the machines could not offer the demanded resources by the migrated SLs. In these cases, more machines (VMs/PMs) could be activated to place these migrated SLs. This can affect migration costs due to the probable increase in migration number and time. We plan to study this aspect in our future work.

7 Discussions and insights

This section is devoted to discussing the major findings of our work and presenting some open discussions and some insights into overheads and scalability.

7.1 Key findings

As per the results presented and analyzed in the previous section, we can argue that the intelligent SL allocator comes with the best results in terms of energy, resource wastage, and costs while combined with our proposed MLC migration strategy. In our scheme, we considered only the actions related to the preselected VMs in each allocation, which allows saving time and memory. Also, determining a list of pre-

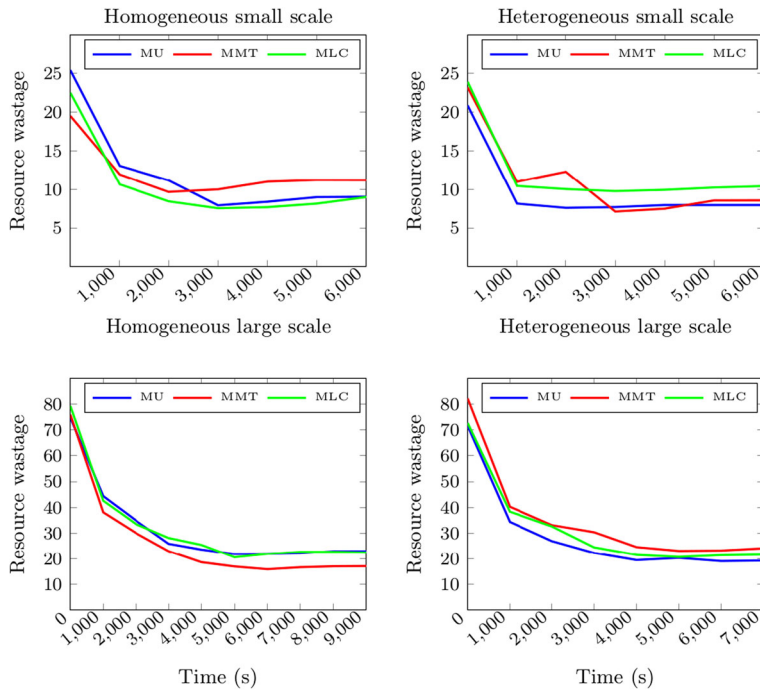


Fig. 12 RL training results of total resource wastage in homogeneous/heterogeneous small/large DC

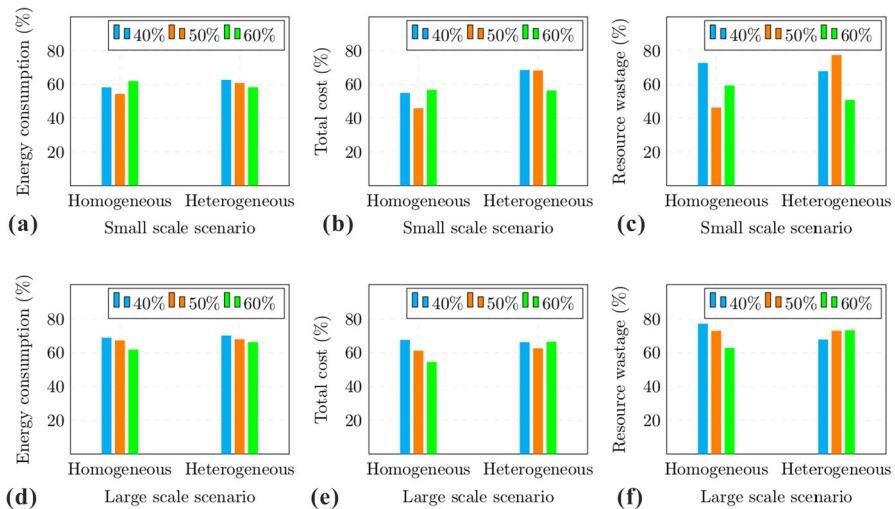


Fig. 13 Effects of UL threshold on the gains in the DC realized by our approach

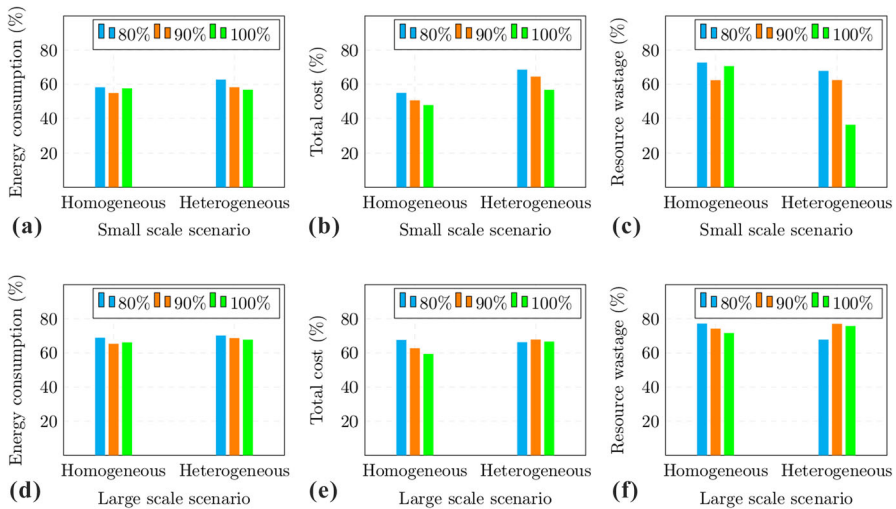


Fig. 14 Effects of OL threshold on the gains in the DC realized by our approach

selected VMs for each SL can be a pain in the ass, especially when the number of SLs is so important. In addition, in some cases, even if the available resources are sufficient to host the migrated SL, if the allowed number of licenses in a given machine is already used, it is not possible to place this SL in that PM. These contradictory constraints can affect the results and limit the consolidation possibilities. Moreover, we noted attempts at non-compliant consolidation which reached 16147 in the small-scale heterogeneous scenario and 215570 in the large-scale homogeneous scenario by the LAB_MLC scheme. These consolidation attempts, if they have been executed will result in huge losses in terms of the cost caused by the non-compliance. Finally, the MLC policy supposes the existence of other instances of software with the same license in the destination VM. If this constraint is not fulfilled, the migration could be more costly due to the fees of the migrated SL.

7.2 Open discussion and limitations of COMPaRL

Today, many organizations are migrated to multi-cloud and hybrid cloud environments implying many deployment modes (private and public cloud, on-premises hardware, Edge systems, etc.) for many reasons like modernization, and managing unforeseen situations such as COVID 19, and cloud bursting to handle peak demands. This transformation creates more complex systems. In a hybrid cloud, public components and private components are integrated and work together due to the managed shared data coming from similar sources. Some application components are orchestrated in the public cloud through micro-services, for example, using Docker container and Kubernetes technologies. To handle this complexity, and for reasons of uniformity and flexibility, we used the same model for both private and public clouds. For private clouds, our approach can be used to optimize energy consumption. In our model, we

considered the energy consumption optimization instead of the energy cost due to temporal and spatial heterogeneity because electricity prices usually vary according to time and geographic location [8]. But, optimizing energy consumption implicitly optimizes energy costs as they are proportional [13] and the latter is based on per unit energy cost.

For public components, the VM cost is optimized for the consumer but also, minimizing the number of VMs offers more opportunities to minimize the number of the hosting PMs, which in turn allows for optimizing the energy consumption as servers are the most consumers in the data center. With the competitiveness of today's multi-cloud environments with the multiplied choices, if the provider can offer more optimized options for the customers this allows stay competitive while optimizing the resource wastage and the energy consumption. Considering the license cost in our model also allows us to avoid non-compliant situations and thus their economic penalties to the provider's profile.

Nevertheless, the major limitation of our strategy is the weight selection of the optimization model due to the heterogeneity of the objectives with different units. Normalization is done before weighting the objective function. But, this is still insufficient to use our model in hybrid clouds, especially given that we used the same cost model for both types of VMs when the costs of running a VM on a private cloud are not fixed per hour but instead deviate on the energy cost. An in-depth study is planned in this regard.

Moreover, we treated here the processor-based metric that allows us to accurately determine the number of cores per physical machine, and therefore the number of licenses in that machine, as in [21]. This is due to the lack of models that describe or formalize the licensing rules in virtualized cloud systems.

In addition, we control the compliance per physical machine based on the total number of licenses. Thus, an allocation is considered non-compliant if the total license number per PM is exceeded independently of how it is counted based on the total number of cores or threads. If each virtual CPU (vCPU) is mapped to only one physical CPU core (depends on the CPU affinities), then our model can be used with public cloud VMs with virtual CPUs and hyperthreading offerings. Otherwise, an extension of our model to consider vCPU offerings is a necessity. This is also encouraged by the fact that VM resizing makes it possible to constrain the number of VM vCPUs to reduce license costs and ensure vertical scalability.

7.3 Discussion of overheads

We discussed in [8] the virtualization overheads when hosting a VM can increase the CPU usage by 11%-17% compared to containerized and traditional methods. This is mostly caused by the use of supplementary software for virtualization. Nevertheless, the security and isolation levels that offer a VM and the scalability and lightness of containers were amongst the first reasons for considering a VM-container virtualization here. This is encouraged by the fact that the overheads generated by the abstraction rules (cgroups and namespaces) for isolation in workload containerization are minimal. Thus, there is no significant performance overhead impact of adding a container

to a hypervisor-based solution in terms of resources. In addition to the virtualization architecture, the generated overhead along the consolidation process is not negligible. The most important part is caused by live migration which introduces energy overhead to the providers due to the copy of memory and the state of the processes. Indeed, if the migration is carried out over a long period of time, it will result in huge resource utilization, which will increase the application execution time and lead to more resource wastage, resulting in performance degradation. The migration cost is affected by the different migration policies for the same combination of migrated objects [8] caused by the migration duration and the additional consumed energy. This increases the overhead of data transmissions. The number of migrations is another factor increasing the migration overhead and causing the unnecessary usage of resources and affecting the application performance. In the current work, we do not consider the above-mentioned overhead components for reasons of simplicity since, to the best of our knowledge, our work is the first to address compliance with the processor-based licensing metric in resource consolidation at the SaaS level. Thus, we believe that our strategy minimizing resource wastage can decrease the generated migration overhead. Also, considering the virtualization overhead, the migration overhead parameters, and minimizing the migration number can minimize the overhead along with the migration process. This will be addressed in future work.

7.4 Scalability discussion and insights

Instead of the generated overhead and costs, live migration allows a flexible and scalable resource allocation [31]. Scalability is one of the benefits of adopting virtualized cloud systems when resources are scaled up and down according to customers' actual needs.

After a scalability test here [14] in terms of execution time while varying the number of software and the number of licenses alternately, we realized that the basic version of the LAB heuristic proved its scalability. In terms of time, CPE is an improvement of the LAB heuristic which, in addition to this latter, performs a simple test to choose the machine having the best power efficiency. While the LAF heuristic is a rapid allocator compared to the two other heuristics. For the intelligent allocator, in the reinforcement learning method, if additional state variables are considered, the state space grows exponentially. This typical dimensionality problem of the RL technique has been addressed in our work by minimizing the number of possible actions for each state thus the number of state-action combinations. Indeed, our agent evaluates the allocation only for the preselected VMs in each time step as stated above. This minimizes not only the computation time needed to determine an optimal policy but also the disk space to store the Q-table updates since its dimensions increase with the state/action spaces. In addition to our strategy, considering a parallel Q-learning can help to further optimize the time to resolve the RL dimensionality challenge. This will be deeply studied in our future work.

Besides that, the scalability of a multi-objective RL-based solution has been proven on VM consolidation cases in [20].

Furthermore, in our previous survey, we discussed some architectural aspects in workload consolidation in general which affect scalability. Here, we proposed a partially distributed architecture to guarantee scalability which is planned to be extended in a fully decentralized solution in the future. We believe that a totally decentralized approach could offer more opportunities for better scalability.

Blockchain technology showed its strengths for resource management in virtualized cloud systems in terms of energy consumption optimization [32] and privacy management [22], among others. We believe that the ledger technology, based on the transfer of value (software licenses in our case), also allows facilitating the task of auditing in a software asset management process (SAM) and manages licenses and their compliance transparently and securely.

Yet, distributing the consolidation decision via a multi-agent system [33] could help parallelize the single RL agent, resolving the dimensionality problem of the RL technique and ensuring better scalability. This allows auto-scaling and a totally automated solution via the self-organization of the agents in addition to the automation offered by the RL solution. Indeed, scaling resources in an optimized and automated manner ensures elasticity [34]. In our work, the optimization process makes it possible to avoid wasting resources and the energy generated thanks to horizontal scalability [34]. This resulted in more efficient use of resources and optimized energy and costs [14]. Our automatic solution, in addition to the elastic management of resources and software licenses, focuses on compliance.

8 Conclusion and outlook

This article proposes an RL-based approach for compliance-aware software license consolidation in commercial cloud environments, namely COMPaRL. The presented and analyzed results showed the efficiency of our scheme when the proposed RL allocator surpasses the heuristic approaches with more important savings in all the optimized objectives regardless of the resource heterogeneity in small and relatively big data centers. Especially, in terms of costs, while combined with our proposed MLC policy, the intelligent approach realized the best results. Moreover, our scheme realized up to 71.38%, 73.49%, 71%, 68.66%, and 77.45% savings in terms of energy consumption, license costs, total costs, VM costs, and resource wastage, respectively. We can conclude that our approach increases the automation of license and resource optimization and ensures elasticity and compliance. It enables resource efficiency, energy savings across the cloud stack and reduces overall costs.

Looking into the other aspects related to the consolidation service, such as migration costs, performance, and affinity, many extensions will be added in future work. The first direction consists to consider other metrics and licensing models. This direction is another dimension of extension to the actual work. The second direction is to implement and test other scalarization functions and a Pareto approach in our future multi-objective RL work instead of the weighted sum. As a third direction, we plan to conduct more tests on other different standard datasets, once possible, in order to definitively confirm the performance and robustness of our proposed model. The fourth direction is to test our approach with other models based on deep reinforcement learn-

ing techniques in order to give academics and practitioners more amplification on how to deal with the problem of Software License Consolidation in Cloud environments. Finally, we believe that, in addition to the options above, a Blockchain-based license management strategy can ensure visibility and help track the real usage of the software licenses in highly virtualized cloud data centers.

We are also motivated to investigate and test the effects of Federated machine learning methods while combined with multi-agent deep reinforcement learning and a decentralized Blockchain-based solution. Further large studies are planned in this respect.

Data availability The dataset generated during the current study is available from the corresponding author on reasonable request.

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

References

1. Vion A-L (2018) Software asset management and cloud computing. (gestion du patrimoine logiciel et cloud computing). PhD thesis, Grenoble Alpes University, France
2. Sen A, Garg A, Verma A, Nayak T (2011) Cloudbridge: On integrated hardware-software consolidation. *ACM SIGMETRICS Performance Evaluation Review* 39:14–25
3. Murthy MKM, Ameen MN, Sanjay HA, Yasser PM (2013) Software licensing models and benefits in cloud environment: A survey. In: *Proceedings of International Conference on Advances in Computing. Advances in Intelligent Systems and Computing*, vol. 174, 645–650. Springer, New Delhi
4. Magnanini F, Ferretti L, Colajanni M (2019) Efficient License Management Based on Smart Contracts Between Software Vendors and Service Providers. In: *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, 1–6. IEEE, Cambridge, MA, USA
5. Höfer CN, Karagiannis G (2011) Cloud computing services: taxonomy and comparison. *Journal of Internet Services and Applications* 2:81–94
6. Oracle: Database Licensing. <https://www.oracle.com/assets/databaselicensing-070584.pdf> (2020)
7. Yousefipour A, Rahmani AM, Jahanshahi M (2018) Energy and cost-aware virtual machine consolidation in cloud computing. *Software Practice and Experience* 48:1758–1774
8. Helali L, Omri MN (2021) A survey of data center consolidation in cloud computing systems. *Computer Science Review* 39:100366
9. Andrae AS, Edler T (2015) On global electricity usage of communication technology: Trends to 2030. *Challenges* 6:117–157
10. Khemili W, Hajlaoui JE, Omri MN (2022) Energy aware fuzzy approach for placement and consolidation in cloud data centers. *Journal of Parallel and Distributed Computing* 161:130–142
11. Mann ZA (2018) Resource optimization across the cloud stack. *IEEE Trans Parallel Distrib Syst* 29:169–182
12. Tchana A, Palma ND, Safieddine I, Hagimont D (2016) Software consolidation as an efficient energy and cost saving solution. *Futur Gener Comput Syst* 58:1–12
13. Khan AA, Zakarya M, Khan R, Rahman IU, Khan M, Khan A.u.R. (2020) An energy, performance efficient resource consolidation scheme for heterogeneous cloud datacenters. *Journal of Network and Computer Applications* 150:102497
14. Helali L, Omri MN (2021) Heuristic-based approach for dynamic consolidation of software licenses in cloud data centers. *International Journal of Intelligent Systems and Applications(IJISA)* 13
15. Helali L, Omri MN (2022) Software license consolidation and resource optimization in container-based virtualized data centers. *J Grid Computing* 20:13
16. Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Futur Gener Comput Syst* 28:755–768

17. Azizi S, Shojafar M, Abawajy J, Buyya R (2021) Grvmp: A greedy randomized algorithm for virtual machine placement in cloud data centers. *IEEE Syst J* 15:2571–2582
18. Hussein MK, Mousa MH, Alqarni MA (2019) A placement architecture for a container as service (caas) in a cloud environment. *J Cloud Comp*
19. Ding W, Luo F, Gu C, Lu H, Zhou Q (2020) Performance-to-power ratio aware resource consolidation framework based on reinforcement learning in cloud data centers. *IEEE Access* 8:15472–15483
20. Qin Y, Wang H, Yi S, Li X, Zhai L (2020) Virtual machine placement based on multi-objective reinforcement learning. *Appl Intell* 50:2370–2383
21. Chevalier A (2021) Optimization of software license placement in the cloud for economical and efficient deployment. PhD thesis, University of Lyon, France
22. Aujla GS, Barati M, Rana O, Dustdar S, Noor A, Llanos JT, Carr M, Marikyan D, Papagiannidis S, Ranjan R (2020) Com-pace: Compliance-aware cloud application engineering using blockchain. *IEEE Internet Comput* 24(5):45–53
23. Barati M, Aujla GS, Llanos JT, Duodu KA, Rana OF, Carr M, Ranjan R (2021) Privacy-aware cloud auditing for gdpr compliance verification in online healthcare. *IEEE Transactions on Industrial Informatics*
24. García J, Iglesias R, Rodríguez MA, Regueiro CV (2019) Directed exploration in black-box optimization for multi-objective reinforcement learning. *International Journal of Information Technology & Decision Making*
25. Hermenier F, Lorca X, Menaud J-M, Muller G, Lawall JL (2009) Entropy: a consolidation manager for clusters. In: *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*
26. Tran CH, Bui TK, Pham TV (2022) Virtual machine migration policy for multi-tier application in cloud computing based on q-learning algorithm. *Computing*
27. Gao Y, Guan H, Qi Z, Hou Y, Liu L (2013) A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *J Comput Syst Sci* 79:1230–1242
28. Zhang R, Chen Y, Dong B, Tian F, Zheng Q (2019) A genetic algorithm-based energy-efficient container placement strategy in caas. *IEEE Access* 7:121360–121373
29. Piraghaj SF (2016) Energy-efficient management of resources in container-based clouds. PhD thesis, The University of Melbourne, Australia
30. Chowdhury MR, Mahmud MR, Rahman RM (2015) Implementation and performance analysis of various vm placement strategies in cloudsim. *Journal of Cloud Computing* 4:20
31. Motaki SE, Yahyaouy A, Gualous H (2021) A prediction-based model for virtual machine live migration monitoring in a cloud datacenter. *Computing* 103:2711–2735
32. Xu C, Wang K, Guo M (2017) Intelligent resource management in blockchain-based cloud datacenters. *IEEE Cloud Computing* 4(6):50–59
33. Helali L, Brahmi Z (2017) Self-organizing agents for dynamic network- and qos-aware service composition in cloud computing. In: *Information Systems Architecture and Technology: Proceedings of 37th International Conference on Information Systems Architecture and Technology – ISAT 2016 – Part II*, 111–124. Springer, Cham
34. Al-Dhuraibi Y, Paraiso F, Djarallah N, Merle P (2018) Elasticity in cloud computing: State of the art and research challenges. *IEEE Trans Serv Comput* 11(2):430–447