Integrating Heuristic and Machine-Learning Methods for Efficient Virtual Machine Allocation in Data Centers

Ali Pahlevan, Xiaoyu Qu, Marina Zapater, *Member, IEEE*, David Atienza, *Fellow, IEEE*

***Abstract*—Modern cloud data centers (DCs) need to tackle efficiently the increasing demand for computing resources and address the energy efficiency challenge. Therefore, it is essential to develop resource provisioning policies that are aware of virtual machine (VM) characteristics, such as CPU utilization and data communication, and applicable in dynamic scenarios. Traditional approaches fall short in terms of flexibility and applicability for large-scale DC scenarios. In this paper we propose a heuristic- and a machine learning (ML)-based VM allocation method and compare them in terms of energy, quality of service (QoS), network traffic, migrations, and scalability for various DC scenarios. Then, we present a novel hyper-heuristic algorithm that exploits the benefits of both methods by dynamically finding the best algorithm, according to a user-defined metric. For optimality assessment, we formulate an integer linear programming (ILP)-based VM allocation method to minimize energy consumption and data communication, which obtains optimal results, but is impractical at runtime. Our results demonstrate that the ML approach provides up to 24% server-to-server network traffic improvement and reduces execution time by up to 480x compared to conventional approaches, for large-scale scenarios. On the contrary, the heuristic outperforms the ML method in terms of energy and network traffic for reduced scenarios. We also show that the heuristic and ML approaches have up to 6% energy consumption overhead compared to ILP-based optimal solution. Our hyper-heuristic integrates the strengths of both the heuristic and the ML methods by selecting the best one during runtime.**

***Index Terms*—cloud data centers, greedy heuristic, machine learning, hyper heuristic, integer linear programming, energy-network traffic trade-offs, QoS, scalability assessment.**

I. Introduction

Cloud computing has recently been brought into focus in both academia and industry due to the increase of applications and services. Consequently, there has been a rapid growth in the number of data centers (DCs) in the world, leading to high energy consumption costs [1]. An approximate power breakdown shows that both information technology (IT) equipments (i.e., server, storage and network) and cooling system encompass over 88% of the total power of a modern DC [2]. Moreover, the explosion of data-intensive applications in current DCs, has led to uneven traffic, bandwidth and communication needs across applications [3].

Virtual machine (VM) consolidation [4] is widely used to minimize energy consumption based on the peak, off-peak, or average CPU usage of VMs [5] to pack them together into the minimum number of servers without degrading quality of service (QoS). However, the dynamic nature of cloud workloads impacts consolidation in two aspects: i) the CPU-load correlation across VMs (i.e, the similarity of CPU utilization traces and the coincidence of their peaks); and ii) the data exchange across VMs (i.e., data correlation) [6]. In this context, several works use heuristics to either address CPU-load correlation, consolidating VMs when their peak utilizations do not coincide [7], [8], or take data correlation into account [9], [10]. Nonetheless, jointly incorporating both metrics in a multi-objective optimization is an important aspect missing from prior works, as studied in a recent survey [11], which significantly increases the complexity of VM allocation.

As complexity raises, integer linear programming (ILP)-based methods become unfeasible at runtime to provide an optimal solution. Similarly, heuristics are problem-specific and less sensitive to dynamic environments, and their benefits become limited for large problems. Thus, when tackling dynamic problems with large state and/or action spaces, machine learning (ML) methods, and in particular reinforcement learning (RL), are suitable techniques [12]. However, in real DC scenarios, VM allocation faces the need to incorporate and assess a wide range of metrics (energy, QoS, network, etc.). This challenges the deployment of ML methods due to their limited configurability. The proposal of methods that balance the trade-offs across these metrics, or dynamically change the optimization goals during run-time to meet DC constraints, remains an open challenge, as it requires a deep assessment on the previous techniques, together with the integration of their strengths. Within this context, hyper-heuristics are a promising solution to leverage the benefits of VM allocation approaches. Hyper-heuristics are *"heuristics that choose heuristics"* [13] and allow to determine which method to use depending on the current DC status, providing better trade-offs than when using the methods separately.

In this paper, we first propose and assess two different approaches to tackle the VM allocation problem: i) a two-phase greedy heuristic, and ii) a ML-based approach. Both approaches exploit CPU-load and data correlations, together with information about DC network topology. Our strategies consolidate VMs into the minimum number of servers and racks, and set dynamic voltage and frequency scaling (DVFS) appropriately. Then, we present a novel hyper-heuristic method that integrates the strengths of both heuristic and ML methods. We evaluate our approaches in terms of energy consumption, QoS degradation, network traffic, number of migrations and scalability, and compare them to an ILP-based optimal solution and state-of-the-art methods.

In particular, our main contributions are as follows:

- We propose a multi-objective hyper-heuristic method to

This work has been partially supported by the YINS RTD project (No. 20NA21_150939), funded by Nano-Tera.ch with Swiss Confederation Financing and scientifically evaluated by SNSF, the EC H2020 MANGO FET-HPC project (Agreement No. 671668), and the ERC Consolidator Grant COMPUSAPIEN (Agreement No. 725657).

The authors are with the Embedded Systems Laboratory (ESL), EPFL, Switzerland e-mail: {ali.pahlevan, xiaoyu.qu, marina.zapater, david.atienza}@epfl.ch.

determine dynamically which method among heuristic and ML is to be used at each time.

- We propose two energy- and network-aware VM allocation methods: i) a two-phase greedy heuristic, and ii) a low-complexity ML-based approach that uses the value iteration algorithm to assign VMs to servers.
- We provide an evaluation of the flexibility, scalability, benefits and drawbacks of heuristic vs. ML methods for the highly dynamic and complex VM allocation problem.
- We compare our proposed solutions with an ILP-based method, that provides an optimal solution, and also with two methods in the state-of-the-art.

Our results show that the heuristic, ML, and hyper-heuristic methods reach almost similar results in terms of energy consumption ($< 2\%$ difference). However, the ML method improves server-to-server network traffic by up to 24% and reduces execution time by up to 480x when compared to conventional approaches for large-scale problems. Also, the hyper-heuristic obtains better trade-offs for different objectives when compared to other approaches.

The remainder of this paper is organized as follows. Sect. II reviews related work. In Sect. III, we describe the system model and used application. Sect. IV provides an overview of the problem description. In Sect. V to VIII we introduce our proposed methods. Sect. IX to XI present the experimental setup and results, followed by conclusion in Sect. XII.

II. Related Work

Research on VM allocation can be generally categorized in energy- and network-aware methods.

A. Energy-aware VM allocation

Regarding energy-aware methods, when deciding the allocation of VMs to physical servers, several works only check that the total size of VMs' load does not exceed the server's capacity [5], [14]. Hence, consolidation solutions are proposed based on per-VM workload characteristics, i.e., the peak, off-peak, and average utilization of VMs [4], [15]. Ahvar *et al.* [16] present a cost and carbon emission-efficient VM placement method and optimize network between DCs, using fuzzy sets. A few studies [7], [17]–[19] consider other VM's attributes, like CPU-load correlation, to achieve further energy savings. Among the latter, Verma *et al.* [17] define VMs' CPU utilization in a time series as a binary sequence where the value becomes '1' when CPU utilization is higher than a threshold. However, this aggressive quantization alters the original behavior and is only applicable when VM envelops are stationary. Meng *et al.* [18] propose a VM sizing technique that pairs two uncorrelated VMs into a super-VM by predicting the workloads. But, once the super-VMs are formed, this solution does not consider dynamic changes, which limits further energy savings. Kim *et al.* [7] present a CPU-load correlation-aware solution based on the First-Fit-Decreasing heuristic to separate CPU-load correlated VMs. The main drawback of this approach is that it cannot be used for online management at large-scale DCs due to its high computational overhead. Lin *et al.* [19] utilize the peak workload characteristics to measure the similarity of VMs' workload. This method achieves better results for VMs whose workload follows a Gaussian distribution.

Ruan *et al.* [20] propose a dynamic migration-based VM allocation method to achieve the optimal balance between server utilization and energy consumption such that all servers operate at the highest performance-to-power levels. Wang *et al.* [21] also address a matching-based VM consolidation mechanism using migration such that active servers can operate close to a desirable utilization threshold. ML is a recently used technique for energy-aware VM allocation in DCs. Farahnakian *et al.* [22] and Masoumzadeh *et al.* [23] present a cooperative multi-agent learning management to minimize the number of active servers managing the overutilized and underutilized servers. Masoumzadeh *et al.* [24] introduce a VM selection task using a fuzzy Q-learning technique to make decisions for migration. Ravi *et al.* [25] also present an energy-efficient Q-learning based technique to decide on VM migrations. The main drawback of those approaches is their high VM migration overhead. Thus, as opposed to short-term decision, Chen *et al.* [8] propose a long-term VM consolidation mechanism such that the total demand of co-located VMs nearly reaches their host capacity during their lifetime period. This algorithm first detects the utilization pattern of each VM based on the four types of simple pulse functions. Then, a heuristic algorithm is used to place all VMs in as few servers as possible. They show a significant reduction in the number of migrations, i.e. only 4% of the total number of VMs, compared to dynamic short-term decision-based methods. Nonetheless, this work ignores the original utilization pattern of the VMs, which is usually a combination of those simple types of functions, achieving lower energy savings. Moreover, none of these approaches consider the data communication between VMs in the allocation process.

B. Network-aware VM allocation

To provide better network resource usage and improve the performance of applications, certain algorithms [16], [26], [27], take into account the communication among VMs in the DC. However, some works assume that data dependencies are given in the form of a directed acyclic graph (DAG). Differently, in practice there are often cyclic communication scenarios, where two VMs regularly exchange information in both directions. As a result, Biran *et al.* [10] propose two new heuristic algorithms to address bidirectional data communication under time-varying traffic demands. However, without the consideration of CPU-load correlation, both approaches are sub-optimal to minimize energy consumption.

III. Data Center Modelling and Applications

In this section we first define the considered DC configuration and describe the used power model. Then, we detail the type of applications tackled in this paper. For the sake of clarity, Table I summarizes the main parameters and notations used throughout this paper.

A. Data center configuration

A DC typically encompasses two main structural components of our interest: i) computing elements –IT– comprising servers and network switches, and ii) cooling systems. We

TABLE I: Overview of the used notation

General Parameters and Variables			
N_s	Total number of servers in DC	$VMcpu$	VMs CPU utilization traces
N_{VM}	Total number of VMs in DC	$VMmem$	VMs memory footprint traces
N_t	Number of samples per time slot	$VMdata$	Data communication demands between VMs
f^{max}	Maximum frequency level of each server	$Freq_j^T$	Selected frequency level of j^{th} server in time slot T
B_{tor}	ToR switch bandwidth	$U_{cpu_j,n}^T$	j^{th} server CPU utilization at n^{th} sample in time slot T
B_{agr}	Aggregation-layer switch bandwidth	$U_{mem_j,n}^T$	j^{th} server memory utilization at n^{th} sample in time slot T
B_{cr}	Core router bandwidth	t_f	Servers' frequency update time during one time slot
C^s	Maximum CPU capacity of each server	t_c	Cooling system update time during one time slot
C^m	Maximum memory capacity of each server		
General Power Model Parameters and Variables			
P_{DC}	DC total power consumption	P_s	Total server power consumption
P_c	Cooling power consumption	P_j	j^{th} server power consumption
P_{IT}	IT power consumption	P_{net}	Total network power consumption
ILP-Based Method Parameters and Variables			
X_j^T	j^{th} server is on ($X_j^T = 1$) or off ($X_j^T = 0$) in T	$D_{j,n}^T$	j^{th} server data communication at n^{th} sample in T
$Place_{j,k}^T$	Whether k^{th} VM is placed on j^{th} server in T	$VMstatus_{j,k,l}$	Placement status of any pair of VMs on j^{th} server
e_j^T	Number of placed VMs on j^{th} server	$BinVMstatus_{j,k,l}$	Whether k^{th} and l^{th} VMs have been allocated to j^{th} server
D_{total}^T	Total data communication amongst servers		
Heuristic Method Parameters and Variables			
$\hat{N}_{server}$	Minimum number of servers to accommodate all VMs	NT_{tor}^r	ToR switch traffic of r^{th} rack
G_{data}	Inter-cluster data communication graph	NT_{agr}^h	Aggregation-layer switch traffic of h^{th} group of racks
ML Method Parameters and Variables			
$\phi_{k,l}^T$	Similarity score between VM_k and VM_l in T	$\phi_{avg,T}^{class}$	Average similarity score among all classes per T
$\rho_{k,l}^T$	Pearson correlation similarity on any pair of VMs	N_{VM}^{server}	Maximum number of VMs can be allocated to each server
$Dist_{k,l}^T$	Euclidean distance between k^{th} and l^{th} VM features	R_j	Total reward value per server
K	Number of classes	λ	Weighting factor to keep reward factors in the same range
N_ω	Number of VMs available in class ω	$\hat{U}_{cpu_j}^T$	Maximum utilization of j^{th} server among samples in T
$\phi_{\omega,T}^{class}$	Per-class (ω) similarity score		
Hyper-Heuristic Method Parameters and Variables			
$\mathbb{O}$	Selected objectives set	$Cost_i$	Cost value per method i
α **and** β	User-defined weighting factors to objectives	Num_i	Number of times that i^{th} method is selected
$\mathbb{M}$	Pool of candidate methods		

consider a typical raised-floor air-cooled DC [28] with 8 racks arranged in hot-cold aisle, as depicted in Fig. 1. Each rack contains 10 to 16 servers, each server with its dedicated power unit, fans and disks. Each rack has one Top-of-Rack (ToR) switch to which all servers in the rack are physically attached, providing a bandwidth B_{tor}. The ToR switch is the lowest layer of a 3-layer tree network topology [14]. Several ToR switches, connect to an aggregate switch, which consolidates traffic into a higher speed link with bandwidth B_{agr}. The Aggregation-layer switches connect to a core router that redirects incoming requests to servers, and tracks and routes VM migrations from one server to another server or DC in another location. The Core router operates in network backbone with the highest speed and bandwidth B_{cr}.

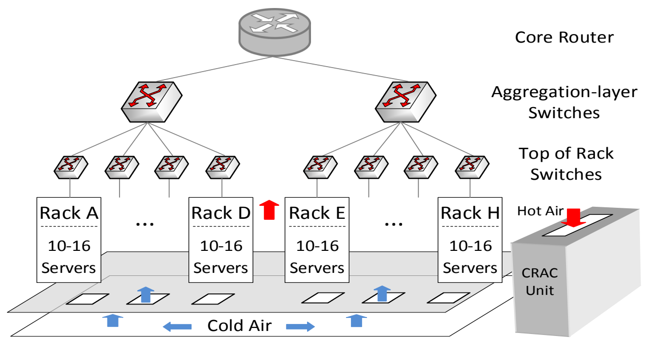

Fig. 1: Considered DC configuration: location of servers, cooling system and multi-layer network topology.

B. Data center power model

In this work, we model total DC power consumption (P_{DC}) as the sum of: i) IT power (P_{IT}), including total server (P_s) and network (P_{net}) power, and ii) the cooling power (P_c):

$$P_{DC} = P_{IT} + P_c \ , \qquad P_{IT} = P_s + P_{net} \tag{1}$$

Server power can be further extended as: $P_s = \sum_{j=1}^{N_s} P_j$, where P_j and N_s specify the j^{th} server power and the total number of servers in the DC, respectively. Following the same methodology than in previous research in the area [1], [29], the major contributors to power consumption in servers are the CPU, memory, fans and disks. Among these, CPU has the largest effect on power, and previous research shows that the power-frequency relation is linear for a given CPU-intensive workload [30]. Hence, server power can be calculated as [29]:

$$P_j = P_{j_{static}} + P_{j_{dyn}}$$
$$\begin{cases} P_{j_{static}} = P_{disk} + P_{fan} + P_{cpu}^{leak} + P_{cpu}^{idle} + P_{mem}^{idle} \\ P_{j_{dyn}} = P_{cpu}^{dyn} \cdot (U_{cpu_j}/100) + P_{mem}^{dyn} \cdot (U_{mem_j}/100) \end{cases} \tag{2}$$

where under $P_{j_{static}}$ we consider all the contributions to power that are workload-independent. P_{disk} and P_{fan} are considered constants for our particular workload, and respectively account for the power consumption of disks and fans. P_{cpu}^{leak} refers to temperature-dependent leakage power. We consider a high fan speed and a low inlet temperature to reduce the effect of temperature-dependent leakage power, considering it as a worst-case constant. P_{cpu}^{idle} and P_{mem}^{idle} are constants that show the idle power consumption of CPU and memory respectively. $P_{j_{dyn}}$ accounts for server dynamic, and is proportional to workload. P_{cpu}^{dyn} and P_{mem}^{dyn} are the fitted constants of our linear model for dynamic CPU and memory power, respectively. Finally, U_{cpu_j} and U_{mem_j} represent CPU and memory utilization of the server, and vary between 0 and 100. The fitted values of P_{cpu}^{dyn} and P_{mem}^{dyn} have been obtained from the models proposed in previous work [1], [29], and assuming that for our workloads (i.e., virtualized banking applications) memory utilization is proportional to the amount of memory accesses.

Network power (P_{net}) is the summation of power of all turned-on switches in each layer of network topology, as [14]:

$$P_{net} = P_{tor} + P_{agr} + P_{cr} \tag{3}$$

where P_{tor}, P_{agr} and P_{cr} denote power consumption of ToR and aggregation-layer switches, and core router respectively.

Finally, to account for cooling power consumption P_c, we use a time-varying Power Usage Effectiveness (PUE) model, proportional to power consumed by IT components; i.e. $P_c = (PUE - 1) \cdot P_{IT}$, as presented in [31]. In this model, PUE mainly depends on the DC room and outside temperature.

C. Applications description

Due to the tight dependency between the nature of the applications and the techniques to be applied, in this paper we consider business-critical applications [32] and, in particular, virtualized banking applications executing batches of tasks. To characterize the power and performance of these applications, we make use of synthetic workloads which are representative of real banking applications according to our industry partners. As realistic CPU usage and memory footprint traces, we use the publicly available traces from Bitbrains, a service provider that provides service to banks such as ING [32]. Bitbrains traces provide data every 5 minutes. Half of the VMs have low variance on CPU usage. However, around 20% of VMs have a very unstable CPU usage. Concerning memory footprint, 80% of VMs use less than 1GB of memory, and in most of them maximum is below 8GB [32].

The Bitbrains traces do not provide any information on the amount of data being exchanged across VMs. Thus, we synthetically generate the amount of data communicated between each pair of VMs using a non-uniform distribution, as detailed in Section IX-A.

IV. Problem Description

In this section we present an overview of the problem description, including the objectives, inputs and outputs. The goal of all the methods proposed in this paper is to minimize the overall server (P_s) and accordingly DC (P_{DC}) power consumption, and network traffic (D_{total}) by means of efficient consolidation-based VM allocation. All the proposed approaches examined consist of two consecutive steps: i) VM characterization and ii) VM allocation, as shown in Fig. 2.

The VM characterization step is used to determine the data communication patterns between VMs, the CPU utilization, and the memory requirements for the next time slot. For the heuristic and ML approach, we use a last-value predictor to

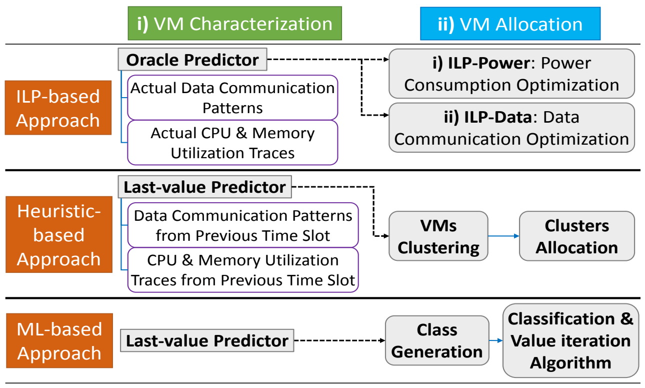
i) VM Characterization
ii) VM Allocation
ILP-based Approach
Oracle Predictor
Actual Data Communication Patterns
Actual CPU & Memory Utilization Traces
i) ILP-Power: Power Consumption Optimization
ii) ILP-Data: Data Communication Optimization
Heuristic-based Approach
Last-value Predictor
Data Communication Patterns from Previous Time Slot
CPU & Memory Utilization Traces from Previous Time Slot
VMs Clustering
Clusters Allocation
ML-based Approach
Last-value Predictor
Class Generation
Classification & Value iteration Algorithm

Fig. 2: Overall diagram of the proposed scenario.

estimate these parameters. The last-value predictor considers that the CPU and memory utilization traces of current time slot (e.g., a time series of n samples, each sample gathered every 5 minutes) are exactly the same than on the previous time slot, as shown in Fig. 3. The specified areas in the figure indicate miss-predictions that can potentially lead to server overutilization and violations, when the predicted CPU utilization is lower than the real one. For the ILP-based method, we assume that, at the beginning of each time slot, all the VM characteristics for the time slot are known (oracle predictor).

The VM Allocation step takes the input VMs CPU, memory, and data communication requirements from the previous step, and the DC network topology. Every time slot, the VM allocation method re-allocates the existing VMs, migrating them if needed to the minimum number of servers such that highly data-correlated VMs are placed together, while highly CPU-load correlated VMs are placed apart. By lowering the number of active servers and racks, unused computing equipment (i.e., idle servers and network switches) can be turned off during that time slot to increase energy efficiency. Turning on/off IT equipment can be applicable to such applications when time slot duration is long enough to prevent significant performance degradation caused by the long transition latency between power modes and changes of resource demands.

After allocating all VMs to the minimum number of servers, the minimum frequency level ($Freq_j^T$) among all the samples in time slot T for each turned-on server is computed as follows:

$$Freq_j^T \geq (U_{cpu_j,n}^T / 100) \cdot f^{max}, \quad \forall n \tag{4}$$

where $U_{cpu_j,n}^T$ indicates the total CPU utilization of the j^{th} server at n^{th} sample in time slot T. In this paper, we assume a homogeneous DC with all servers of the same type. Therefore, f^{max} is equal for all servers and determines the maximum

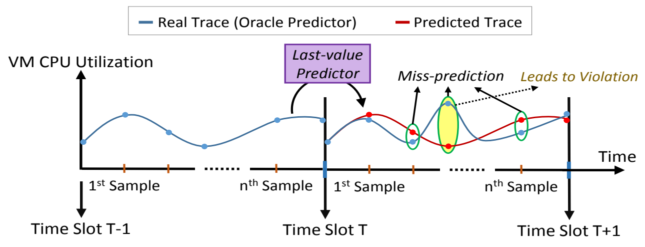

Fig. 3: Time slot and sample description.

frequency level. This equation guarantees that the selected frequency is sufficient to avoid server overutilization for all the samples in time slot T. Thus, in the proposed scenario, violations can only occur due to miss-predictions on the VM usage (e.g., for the case highlighted in Fig. 3), when the server utilization needs a higher frequency than the one selected. In this sense, the ILP method will never exhibit violations, as it uses an oracle predictor. After allocation, for all the proposed methods during one time slot, we update the servers frequency every t_f by computing the maximum server utilization occurred in the previous t_f period. Finally, after computing the total IT power consumption, we update and compute the cooling system power every t_c during one time slot, based on the DC room and outside temperatures.

V. Proposed ILP-based Optimization Method

The ILP method can be divided in two minimization problems: i) power consumption (ILP-Power), and ii) data communication (ILP-Data). Due to the varying nature of DC workloads (i.e., VMs' utilization patterns change over time), there is an optimal VM allocation for each time slot and, thus, a need to invoke the ILP-based method. The time slot duration is a parameter that can be adjusted by the DC operator depending on the granularity of the traces used, to increase accuracy. We define the ILP formulations in a generic form regardless of the network topology constraints. Hence, an optimal server-to-server data communication is obtained that represents the total network traffic. In the following subsections, we describe these two optimization goals in detail.

A. Power consumption optimization

The proposed ILP-Power VM allocation aims to minimize overall server power based on the Eq. 2. The minimization objective is given by Eq. 5, where P_s^T and $P_{j,n}^T$ denote overall and j^{th} server power consumption at the n^{th} sample of the T^{th} time slot, respectively. N_s and N_t are the number of servers in the DC and number of samples in one time slot, respectively. The binary variable X_j^T is defined to indicate whether the j^{th} server is on ($X_j^T = 1$) or off ($X_j^T = 0$) in the T^{th} time slot.

We use the binary variable $Place_{j,k}^T$ to indicate whether k^{th} VM ($k \in 1,2,...,N_{VM}$) is placed on j^{th} server in T^{th} time slot. N_{VM} is the total number of VMs available in the DC. Matrices $VMcpu_{k,n}^T$ and $VMmem_{k,n}^T$, contain the k^{th} VM's CPU utilization and memory footprint at n^{th} sample, respectively, during the T^{th} time slot. Similarly, $U_{mem_j,n}^T$ indicates the total memory utilization of the j^{th} server.

The minimization problem is subject to the constraints given in Eq. 8 to 12. Constraint 1 forces each VM to be placed only in one server. Constraints 2 and 3 enforce that aggregated CPU and memory utilizations of the VMs in j^{th} server do not exceed the maximum CPU and memory capacities, i.e. C^s and C^m, respectively. In Constraint 4, the integer variable e_j^T is used to specify the number of placed VMs on the j^{th} server. This value is upper-bounded by N_{VM}. Constraint 5 guarantees that if no running VMs are placed on j^{th} server in the T^{th} time slot ($e_j^T = 0$), this server can be turned off ($X_j^T = 0$).

Server power in the objective function (Eq. 5) should be written as $P_j^T = X_j^T \cdot (P_{j_{static}}^T + P_{j_{dyn}}^T)$. However, this would introduce a non-linearity in the ILP problem (due to the product of variables $X_j^T \cdot U_{cpu_j}^T$ and $X_j^T \cdot U_{mem_j}^T$ in $P_{j_{dyn}}^T$). Constraint 5 avoids this issue as, when the number of VMs on j^{th} server (e_j^T) as well as the server CPU and memory utilization is zero, $X_j^T = 0$. On the contrary, if $1 \le e_j \le N_{VM}$, then $X_j^T = 1$. Therefore, we can write: $X_j^T \cdot U_{cpu_j,n}^T = U_{cpu_j,n}^T$ and $X_j^T \cdot U_{mem_j,n}^T = U_{mem_j,n}^T$.

$$\begin{aligned}
\min \quad P_s^T &= \sum_{j=1}^{N_s} \sum_{n=1}^{N_t} X_j^T \cdot P_{j,n}^T \\
&= \sum_{j=1}^{N_s} \sum_{n=1}^{N_t} (X_j^T \cdot P_{j_{static}} + P_{j_{dyn},n}^T) \\
&= \sum_{j=1}^{N_s} \sum_{n=1}^{N_t} (X_j^T \cdot P_{j_{static}} + P_{cpu}^{dyn} \cdot U_{cpu_j,n}^T + \\
&\qquad P_{mem}^{dyn} \cdot U_{mem_j,n}^T)
\end{aligned} \tag{5}$$

where

$$U_{cpu_j,n}^T = \sum_{k=1}^{N_{VM}} Place_{j,k}^T . VMcpu_{k,n}^T \tag{6}$$

$$U_{mem_j,n}^T = \sum_{k=1}^{N_{VM}} Place_{j,k}^T . VMmem_{k,n}^T \tag{7}$$

subject to the following Constraints:

$$1.\ \sum_{j=1}^{N_s} Place_{j,k}^T = 1 \tag{8}$$

$$2.\ U_{cpu_j,n}^T \le C^s \tag{9}$$

$$3.\ U_{mem_j,n}^T \le C^m \tag{10}$$

$$4.\ e_j^T = \sum_{k=1}^{N_{VM}} Place_{j,k}^T \tag{11}$$

$$5.\ X_j^T \ \le \ e_j^T \ \le \ X_j^T N_{VM} \tag{12}$$

B. Data communication optimization

The amount of data exchanged between VMs directly impacts network traffic and response time. In practice, two VMs regularly exchange a varying amount of data. Our goal is to minimize total data communication (network traffic, D_{total}^T) amongst the servers. In our formulation, $D_{j,n}^T$ represents the j^{th} server data communication; i.e., the amount of data transferred by a server at the n^{th} sample of the T^{th} time slot.

To express $D_{j,n}^T$, the binary variable $BinVMstatus_{j,k,l}^T$ indicates whether both k^{th} and l^{th} VMs have been allocated to j^{th} server ($BinVMstatus_{j,k,l}^T = 0$); otherwise, $BinVMstatus_{j,k,l}^T = 1$ in the T^{th} time slot. The matrix $VMdata_{k,l,n}^T$ contains the amount of data transferred from the k^{th} to l^{th} VM at the n^{th} sample during T^{th} time slot.

The constraints of the problem are formulated in Equations 15 to 19, as follows. Constraints 1, 2 and 3 are the same as those of ILP-Power. Constraint 4 determines the status of the k^{th} and l^{th} VMs on the j^{th} server. The variable $VMstatus_{j,k,l}^T$ is used to specify the status of any pair of VMs based on the status of each VM on j^{th} server. As this variable is the sum of two binary variables, it can take only three different values (i.e. 0, 1, and 2): i) the k^{th} and l^{th} VMs are allocated to the j^{th} server ($Place_{j,k}^T = 1$ & $Place_{j,l}^T = 1$), then $VMstatus_{j,k,l}^T = 0$; or ii) either the k^{th} or l^{th} VM is allocated to the j^{th} server ($Place_{j,k}^T = 1$ & $Place_{j,l}^T = 0$ or vice versa), then $VMstatus_{j,k,l}^T = 1$; or iii) neither the k^{th} and l^{th} VMs are allocated to the j^{th} server ($Place_{j,k}^T = 0$ & $Place_{j,l}^T = 0$), $VMstatus_{j,k,l}^T = 2$.

The original data communication objective is written as $D_{j,n} = \sum_{k=1}^{N_{VM}} Place_{j,k} \cdot \sum_{l=1,\ l\neq k}^{N_{VM}} (1 - Place_{j,l}) \cdot VMdata_{k,l,n}$. To remove the non-linearity in the equation, constraint 5 is used to compute per-server data communication and demonstrates that, if $VMstatus_{j,k,l} = 0$, then $BinVMstatus_{j,k,l} = 0$, and '1' otherwise. In other words, if both VMs are allocated to the same server, then $BinVMstatus_{j,k,l} = 0$.

$$\min \quad D^T_{total} = \sum_{j=1}^{N_s} \sum_{n=1}^{N_t} D^T_{j,n} \tag{13}$$

where

$$D^T_{j,n} = \sum_{k=1}^{N_{VM}} \sum_{\substack{l=1 \\ l\neq k}}^{N_{VM}} [Place^T_{j,k} - (1 - BinVMstatus^T_{j,k,l})] \cdot VMdata^T_{k,l,n} \tag{14}$$

subject to the following Constraints:

$$1.\ \sum_{j=1}^{N_s} Place^T_{j,k} = 1 \tag{15}$$

$$2.\ U^T_{cpu_{j},n} \leq C^s \tag{16}$$

$$3.\ U^T_{mem_{j},n} \leq C^m \tag{17}$$

$$4.\ VMstatus^T_{j,k,l} = (1 - Place^T_{j,k}) + (1 - Place^T_{j,l}) \tag{18}$$

$$5.\ BinVMstatus^T_{j,k,l} \leq VMstatus^T_{j,k,l} \leq 2 \cdot BinVMstatus^T_{j,k,l} \tag{19}$$

VI. Proposed Two-Phase Greedy Heuristic Method

In this section we propose a two-phase greedy heuristic algorithm (Heuristic, in what follows) that, at each time slot T, jointly minimizes power consumption –P^T_s– and network traffic –D^T_{total}– (Phase 1), and then allocates resulting traffic in a network topology-aware fashion (Phase 2).

A. Phase 1 - VM Clustering

We split this phase in two steps and use a method similar to the one presented in [6]. First, at time slot T, all VMs available in the system are represented as points in a two dimensional (2D) plane. Based on the data and CPU-load correlation properties, as highly data-correlated VMs should be clustered together while highly CPU-load correlated VMs should be placed apart, a function is defined to calculate attraction and repulsion forces between each two VMs. Nonetheless, differently from the original algorithm [6], we calculate the attraction force as a worst-case peak bidirectional data exchanged between each two VMs during the time slot. Similarly, the repulsion force is computed as a worst-case peak CPU utilization when the peaks of two VMs coincide during the last time slot. As a result, the points are remapped in the 2D plane with new coordinates based on the computed forces.

In the second step, after finding the final position of the VMs, we determine the minimum number of clusters (i.e., servers), $\hat{N}_{server}$, as follows:

$$\hat{N}_{server} = max\{\hat{N}^{cpu}_{server}, \hat{N}^{mem}_{server}\}$$
$$\begin{cases} \hat{N}^{cpu}_{server} = max_n(\sum_{k=1}^{N_{VM}} VMcpu^{T-1}_{k,n}/C^s) \\ \hat{N}^{mem}_{server} = max_n(\sum_{k=1}^{N_{VM}} VMmem^{T-1}_{k,n}/C^m) \end{cases} \tag{20}$$

where $\hat{N}^{cpu}_{server}$ and $\hat{N}^{mem}_{server}$ denote the minimum number of servers needed to comply with the VMs' CPU and memory utilization requirements, respectively. Hence, $\hat{N}_{server}$ is equal to the minimum number of servers to accommodate all the VMs while satisfying both the VMs CPU and memory utilization.

Then, we utilize a modified version of the k-means algorithm [6] to cluster VMs with respect to the distance between two VMs obtained from the repulsion and attraction phase in the 2D plane. Differently from the original k-means algorithm, we define a cap per cluster (i.e., C^s and C^m) when considering the VMs' CPU and memory utilization. Moreover, the initial centroid of clusters are not set randomly, but instead calculated based on the last position of points available in the previous time slot. We start with the minimum number of clusters, $\hat{N}_{server}$, to allocate the VMs to the clusters with shortest distance. If unfeasible, we increment the number of clusters by one. The process is iterated until all VMs are allocated to the minimum number of possible clusters. Our method guarantees that the total load of each cluster at each sample does not exceed C^s and C^m during the time slot. However, violation occurs due to miss-predictions, leading to delays in workload execution and, eventually, to their execution in the next time slot (with 100% prediction accuracy, no violation occurs).

B. Phase 2 - Clusters Allocation

In this phase, we allocate the clusters to the appropriate servers considering the DC network structure as described in Algorithm 1. This algorithm fills up the racks one by one, reducing the number of active switches, and minimizing network power, while keeping highly-communicating servers close to each other.

Algorithm 1 Cluster Allocation

```
Input: Network topology and data communication graph ($G_{data} = \{V, E\}$)
    $V$ = clusters & $W(E)$ = data communication between clusters
Output: Cluster allocation
 1: $Edge^{max} \leftarrow$ Max. edge ($W(E_{w,z}) + W(E_{z,w})$) between any $V_w$ and $V_z$
 2: $NT^h_{agr} \leftarrow 0$   Initial network traffic of $h^{th}$ agr. switch
 3: for $r = 1$ : Total racks do
 4:     $NT^r_{tor} \leftarrow 0$   Initial ToR switch network traffic of $r^{th}$ rack
 5:     Unused servers of $r^{th}$ rack $\leftarrow$ Total servers of $r^{th}$ rack
 6:     while (Selected clusters $\leq$ Unused servers of $r^{th}$ rack) & ($NT^r_{tor} \leq B_{tor}$)
        & ($NT^h_{agr} \leq B_{agr}$) do
 7:         if $V_w$ and $V_z$ have not been allocated then
 8:             Allocate clusters $w$ and $z$ to two servers in $r^{th}$ rack
 9:             $NT^r_{tor} \leftarrow$ Update traffic of servers of $r^{th}$ rack
10:             $NT^h_{agr} \leftarrow$ Update traffic of racks of $h^{th}$ group
11:             Update unused servers in $r^{th}$ rack
12:         else if $V_w$ or $V_z$ has not been allocated then
13:             Allocate $w$ or $z$ to one server in $r^{th}$ rack
14:             $NT^r_{tor} \leftarrow$ Update traffic of servers of $r^{th}$ rack
15:             $NT^h_{agr} \leftarrow$ Update traffic of racks of $h^{th}$ group
16:             Update unused servers in $r^{th}$ rack
17:         end if
18:         Combine $V_w$ and $V_z$
19:         Update $W(E)$
20:         if All clusters allocated then
21:             Terminate
22:         end if
23:         $Edge^{max} \leftarrow$ Find maximum edge weight
24:         Find number of selected clusters ('1' or '2') when both clusters
            have not been allocated
25:     end while
26: end for
```

The output of the modified k-means creates an edge-weighted data communication graph ($G_{data} = \{V, E\}$), where set V and $W(E)$ represent the clusters and the amount of data transferred across clusters, respectively. The algorithm first selects the maximum edge weight ($Edge^{max}$) and initializes the amount of traffic transferred through all aggregation-layer switches (NT^{h}_{agr}) for different groups of racks (lines 1 and 2). Then, we select the first rack and try to fill it up. For the selected rack (r^{th} rack), we first initialize its ToR switch traffic (NT^{r}_{tor}) and the number of unused servers with the total number of servers available in r^{th} rack (lines 4 and 5). Then, for clusters related to the selected edge (lines 6 $\sim$ 25) if either: i) two clusters have not been allocated yet and the number of unallocated clusters is less than the unused servers in the rack, ii) NT^{r}_{tor} of the new selected clusters is less than the B_{tor}, and iii) NT^{h}_{agr} related to the selected rack is less than B_{agr}, we allocate clusters to the servers in that rack. We also update the NT^{r}_{tor}, NT^{h}_{agr}, and the number of unused servers of the r^{th} rack (lines 7 $\sim$ 17). After allocation, we combine clusters V_w and V_z and update $W(E)$ (lines 18 and 19). We repeat until violating those conditions for the rack, and then we select the next rack. This algorithm iterates until all clusters are allocated to physical servers, which guarantees that the bandwidth of switches is not exceeded.

VII. Proposed Machine Learning Method

This section describes a two-step ML approach to allocate VMs to servers. First, we generate offline different classes using k-means according to the features extracted from the VMs' CPU utilization traces. To decide the appropriate number of classes (K), we use a heuristic-based process, as explained in Section VII-A. Second, at runtime, we classify VMs into classes by determining the shortest euclidean distance to each class centroid, and then we use the *value iteration* algorithm, amongst the various RL methods, to allocate the VMs to physical servers. RL is particularly useful in problems with large state and/or action spaces that change dynamically over time and depend on the environment [12]. We use the first week of Bitbrains traces for class generation and for the exploration phase of the ML approach, and the second week of traces for run-time VMs classification and exploitation phase.

A. Class generation – Offline pattern detection

Class generation significantly simplifies the process of VM allocation, reducing the complexity of the value iteration algorithm. In the Bitbrains traces we observe a high-variability but also a daily periodicity in the CPU utilization traces, making them suitable for classification. As memory resources are more over-provisioned than CPU resources, and less critical, our class generation only takes into account the CPU utilization traces to generate classes. In this step, based on the time slot duration (i.e., 1 hour), we consider each VM's CPU utilization trace per time slot as an individual pattern composed of 12 samples (1 every 5 minutes).

Feature extraction is a key point that greatly affects classification results. In this work, we select a list of features as:

- Maximum and minimum CPU utilization, to represent the range of variation and the absolute value of the traces.
- Time at which the maximum utilization happens, to enable CPU-load correlation techniques.
- Variance, as it shows the trace variability.
- Median, to account for typical values.
- Skewness, which is a measure of the trace asymmetry.
- Kurtosis, which provides an insight on the trace shape.

Then, we use the k-means method to classify CPU utilization patterns [33]. As k-means does not decide on the number of classes (K), we propose the following heuristic.

Heuristic-based process for determining the appropriate number of classes (K): First, we define a similarity score ($\phi^{T}_{k,l}$ in Eq. 21) that expresses the similarity between any pair of VMs, VM_k and VM_l during time slot T. To define this metric we use the Pearson Correlation ($\rho^{T}_{k,l}$), which is effective to judge the similarity on the shape of the traces. However, as the Pearson Correlation cannot reflect the absolute CPU utilization value, we incorporate the euclidean distance ($Dist^{T}_{k,l}$) over all the samples into the metric. As a result, Eq. 21 demonstrates that $\phi^{T}_{k,l}$ is high when two traces have both the same shape and CPU utilization absolute value. Since two VM traces may be totally the same, we consider $(Dist^{T}_{k,l}+1)$ in the denominator to avoid having infinite value when $Dist^{T}_{k,l}=0$, and we normalize the values to $[0,1]$.

$$\phi^{T}_{k,l} = \frac{\rho^{T}_{k,l}}{Dist^{T}_{k,l}+1}\ , \quad Dist^{T}_{k,l} = \|VMcpu^{T}_{k} - VMcpu^{T}_{l}\|_2 \tag{21}$$

Second, for each time slot in one week, we classify the VMs into K different classes, according to their euclidean distance (exhaustively testing different values of K). After classification, we compute the per-class similarity score during each time slot ($\phi^{class}_{\omega,T}$), as calculated by Eq. 22, where N_ω is the number of VMs available in class ω, obtaining K different scores (i.e., as many as classes). We average these K scores to obtain an average similarity score ($\phi^{class}_{avg,T}$) per T.

$$\phi^{class}_{\omega,T} = \frac{\sum_{k=1}^{N_\omega}\sum_{l=1,\ l\neq k}^{N_\omega}\phi^{T}_{k,l}}{N_\omega * (N_\omega - 1)} \tag{22}$$

By increasing the number of classes (K), the similarity score exhibits a logarithmic growth, achieving its highest value of '1', when we have as many classes as patterns. As the number of classes directly impacts the execution time of the ML algorithm, but a high similarity is needed to achieve good accuracy, we heuristically choose a similarity score of 0.5, which leads to 150 classes (K). Because managing 150 classes is still unfeasible for the ML algorithm, we first combine the centroids of classes (i.e., compute the mean of centroids that are below a certain euclidean distance). Then, we delete those classes whose number of VMs are below a certain threshold. By appropriately setting these thresholds, our method allows to reduce the number of classes from 150 to 27, i.e. $K=27$, without decreasing average similarity.

B. Run-time classification and value iteration algorithm

At runtime, we use the second week of traces to classify VMs into the classes resulting from the previous step in each time slot. First, we use the last-value predictor to obtain the last VMs' patterns and extract their features. Then, we assign the VM to the class which has the shortest euclidean distance to the centroid. Finally, we use a RL technique, the value-iteration algorithm, to allocate the VMs to physical servers. Typically, RL models are composed of an agent and

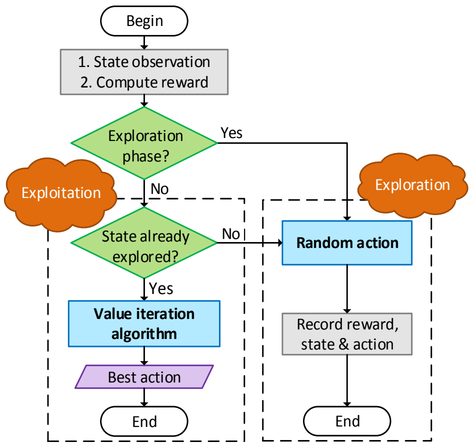

Fig. 4: The overall process of our ML approach.

an environment with a finite set of actions (A) and a state space (S). In the environment, the states are observed and the actions determined by agent are applied. The agent maps actions to states at any decision time. There is a reward function used for each state-action pair (R) which should be maximized by the agent. $R(s,a,s')$, thus, shows the immediate reward value obtained after performing action a in current state s, representing the next state (s') reward value [12].

The proposed ML approach consists of two phases, namely exploration and exploitation, as shown in Fig. 4. If the current state has not been previously explored, we are in the exploration phase, and the agent randomly chooses new actions for the new states, and records the new states and rewards obtained from each action separately. On the contrary, if a state has already been explored, we proceed to the exploitation phase, and use the value iteration algorithm to find the best action among the pool of actions obtained during exploration to maximize the reward. The essential idea is: if we knew the true value of each state, we would simply choose the action that maximizes the expected reward. In our case, at each communication between agent and environment, we apply a set of actions to environment (allocating only one VM to each server). Hence, we don't initially know the state's true value; we only know its immediate reward. For example, a state might have low initial reward but be on the path to a high-reward state. Value-iteration progressively evaluates the states until the solution converges. Nonetheless, it assumes that the agent has a great knowledge about the reward of all states in the environment. Because in a consolidation problem the reward can be clearly stated (i.e., the higher the server utilization, the higher the reward is), the value-iteration method is suitable to solve this problem.

Because the value iteration algorithm uses a vector of actions, the amount of servers active per time slot needs to be defined before VM allocation. To this end, we first compute the maximum number of VMs than can be allocated to each server (N_{VM}^{server}) and the minimum number of needed servers, i.e., $\hat{N}_{server}$, per time slot. Then, the agent decides the VM placement by choosing which VM class should be allocated to each server. If the selected class has no VMs, it picks a VM from the class with a shortest euclidean distance to it. After applying the actions, we update the utilization of each server, set its state and reward, and send them back to the agent. We iterate this communication until all the VMs are allocated.

Finally, to minimize traffic through the different switches considering bandwidth constraints, we fill up the racks one after the other with the servers that have higher data communication (after allocation, we sort the amount of data transferred between any pair of servers in descending order). The following subsections describe the state, actions, and the proposed reward function for the value-iteration algorithm.

1) State and action definitions

Each time slot, the agent provides one action per server to the environment. This means that $\hat{N}_{server}$ actions are applied to $\hat{N}_{server}$ servers. The number of communication steps in one time slot is the maximum number of VMs allowed to be allocated to one server, i.e. N_{VM}^{server}.

The state of each server is defined based on the number of VMs allocated to it and the server utilization. In general, we use three parameters to reflect one state value. Each parameter can get a discrete value in a finite range, as shown in Table II. The first parameter, P1, indicates the number of VMs allocated to server. P2 exhibits the maximum aggregated CPU utilization of co-located VMs during the time slot. We discretize server utilization (from 0 to 100%) in 11 uniform levels. We map the predicted server utilization to the nearest higher level to minimize violations due to miss-predictions. P3 is a binary parameter that shows that if the server is active. The allocation of a VM from one class to a server is considered as an action. We choose the first VM available from the selected class. If the class is empty, we choose a VM from the non-empty class which has the minimum distance to that class.

2) Reward function

Each pair of state-action reward (next state reward) is defined per server ($R_j(s,a,s')$ for jth server; we simply name it R_j) according to two parameters: i) the gap between the current and the maximum utilization (i.e. utilization gap, R_g^j) and, ii) the amount of data transferred by the server (R_d^j) as:

$$R_j = R_g^j - \frac{R_d^j}{\lambda} \tag{23}$$

where λ is a weighting factor used to keep the reward factors in the same range.

Given that we are using consolidation as a strategy for power minimization, we could have three different situations: i) fully utilized server (i.e., $= C^s$), ii) underutilized server ($< C^s$), and iii) overutilized server ($> C^s$). To minimize power consumption via consolidation, R_g^j needs to be highest when the server is fully utilized (we choose 1000). Hence, it is enough to set lower reward values for the rest of situations. For underutilization situations, we choose a positive proportional range between 1 and $C^s/10$, i.e., the higher the server utilization, the higher the reward is. On the other hand, to

TABLE II: State definition (s) and value per server

Parameter	Definition	Range
$P1$	Number of allocated VMs	$0-N_{VM}^{server}$
$P2$	Utilization of server	$0-11$
$P3$	Active/inactive in time slot	$0/1$

minimize violations and QoS degradation, we need to avoid surpassing maximum utilization (C^s). For this purpose, due to the higher importance of server overutilization compared to underutilized server situations, it is enough to choose a lower value to decrease the server reward. Therefore, we set a negative value to utilization above C^s (we choose -1). In practice, any other negative value works, and gives the same results. Thus, R_g^j can be computed as follows:

$$R_g^j = \begin{cases} \frac{C^s}{C^s - \hat{U}_{cpu_j}^T} & \hat{U}_{cpu_j}^T < C^s \\ 1000 & \hat{U}_{cpu_j}^T = C^s \\ -1 & \hat{U}_{cpu_j}^T > C^s \end{cases} \tag{24}$$

where $\hat{U}_{cpu_j}^T$ represents the maximum utilization of j^{th} server among all the samples in the T^{th} time slot, i.e., $max_n(U_{cpu_j,n}^T)$.

Similarly, R_d^j represents the total amount of data transferred by j^{th} server, as follows:

$$R_d^j = \sum_{n=1}^{N_t} D_{j,n}^T = \sum_{k=1}^{N_{VM}} \sum_{l=1}^{N_{VM}} \sum_{n=1}^{N_t} VMdata_{k,l,n}^T \\ VM_k \in server_j \ \& \ VM_l \notin server_j \tag{25}$$

The reward function is computed per-server, aiming to maximize server utilization while minimizing the amount of data should be exchanged between servers.

VIII. Proposed Hyper-Heuristic Method

In this section we present a hyper-heuristic algorithm to dynamically determine which method, among Heuristic and ML, should be used at each time slot T to achieve a specific trade-off across the different objectives. The proposed hyper-heuristic relies on the long-term periodicity of the workloads being executed, and learns the performance of the methods over time. The considered trade-offs (objective set $\mathbb{O}$) are power consumption (P_{DC}), worst-case server overutilization (WCV), and total network traffic of ToR (TN_{tor}), aggregation- (TN_{agr}), and core-layer (TN_{cr}) switches, as the most important metrics from the DC providers perspective. They are computed for each method i in a cost function ($CostFunction(\mathbb{O}_i)$) as:

$$Cost_i = \alpha_1 P_{DC} + \alpha_2 WCV + \alpha_3(\beta_1 TN_{tor} + \beta_2 TN_{agr} + \\ \beta_3 TN_{cr}) \ , \quad \sum_{j=1}^{3} \alpha_j = 1 \ \& \ \sum_{k=1}^{3} \beta_k = 1 \tag{26}$$

where α_j and β_k are user-defined weighting factors that need to be set with respect to the importance that the user gives to a specific objective (each normalized to (0,1]), and whose value can be changed during runtime. In this work, for the sake of clarity, we decided to give the same priority to all objectives (i.e., $\alpha_1 = \alpha_2 = \alpha_3 = 1/3$). Due to the communication distance, we consider higher weight for upper network layers ($\beta_1 = 0.1$, $\beta_2 = 0.3$, and $\beta_3 = 0.6$). Thus, the lower the power consumption, overutilization and network traffic, the lower the cost value is. The proposed algorithm is as follows.

At the beginning of each time slot, one of the methods in the pool of candidate methods ($\mathbb{M}$) (i.e., Heuristic and ML in our case) is selected. To choose which algorithm to be executed, our proposed hyper-heuristic builds a history of the performance of the Heuristic and ML methods by using the cost function (Eq. 26). As described in Algorithm 2, at the beginning of time slot T, we create a hash code of the previous execution ($T-1$) that is stored in a hash table ($HashTable$). For the first time slot, $HashTable$ is empty and we randomly select one of the methods. We generate the hash using the function $HashGenerator$ (line 1) that creates a binary string with the length of the selected objectives ($\mathbb{O}$), in which each character is '1' if the ML performed better in that objective than the Heuristic, and is '0' otherwise. For example, hash code "11011" shows that ML has the best results for four objectives with respect to Heuristic. In $HashTable$, for each observed hash, we also store two entries per method including the cost value ($Cost_i^{Hash}$, $i \in \mathbb{M}$) and how many times we selected that method in the past (Num_i^{Hash}, $i \in \mathbb{M}$). After generating the hash, the algorithm checks in $HashTable$ whether the hash had been already observed. If it exists, we select the method with the minimum $Cost_i^{Hash}/Num_i^{Hash}$ (line 4). Otherwise, we select the method with minimum cost value for $T-1$, and we record only the observed hash (lines 6 and 7). We update the hash entries, i.e. $Cost^{Hash}$ and Num^{Hash}, at the end of T as follows.

After executing the selected method, at the end of time slot T, we collect the results ($\mathbb{O}^T$) per method (lines 9 and 10). Then, we just update the entries of the method with the minimum cost value for corresponding hash (lines 13 and 14).

Algorithm 2 Hyper-heuristic Algorithm

```
Input: $\mathbb{O}_i^{T-1} = \{P_{DC}^{T-1}, WCV^{T-1}, TN_{tor}^{T-1}, TN_{agr}^{T-1}, TN_{cr}^{T-1}\}$, $i \in \mathbb{M}$
       $HashTable$
Output: Select one method from $\mathbb{M}$
 1: $Hash \leftarrow$ HashGenerator($\mathbb{O}_{\mathbb{M}}^{T-1}$)
 2: $HashObserved \leftarrow$ IsHashObserverd($Hash$, $HashTable$)
 3: if $HashObserved == True$ then
 4:     $m \leftarrow$ Select method with $min(Cost_i^{Hash}/Num_i^{Hash})$, $i \in \mathbb{M}$
 5: else if $HashObserved == False$ then
 6:     Record $Hash$ in $HashTable$
 7:     $m \leftarrow$ Select method with minimum CostFunction($\mathbb{O}^{T-1}$)
 8: end if
 9: Execute method $m$ for time slot $T$
10: Obtain $\mathbb{O}_i^T$ at the end of time slot $T$, $\forall i$
11: $Cost_i \leftarrow$ CostFunction($\mathbb{O}_i^T$), $\forall i$
12: $m \leftarrow$ Find method with $min(Cost)$
13: $Cost_m^{Hash} \leftarrow Cost_m^{Hash} + Cost_m$
14: $Num_m^{Hash} \leftarrow Num_m^{Hash} + 1$
```

IX. Experimental Setup and Scenarios

In this section we present the experimental setup and introduce two scenarios to compare the proposed methods.

A. Experimental Setup

1) Data center configuration: We consider two rows of racks in the DC. Each row consists of four 42U racks, and each rack has ten servers. We target Intel S2600GZ servers consisting of 6-core CPU (Intel E5-2620), 9 frequency levels varying from 1.3 to 2.4GHz, and 32GB of memory.

The server power consumption is modeled as in Section III-B; each server consumes constant 16W and 27.2W for disk (P_{disk}) and cooling fan (P_{fan}), respectively. We consider a high fan speed (8000rpm) and a low inlet temperature (22°C) to reduce the effect of temperature-dependent leakage power. Under this condition, leakage power (P_{cpu}^{leak}) is almost constant and 3.1W in the worst-case. Idle power for CPU (P_{cpu}^{idle}) and memory (P_{mem}^{idle}) are 50W and 4W, respectively. The dynamic power of CPU (P_{cpu}^{dyn}) and memory (P_{mem}^{dyn}) are $42.5W$ and $56W$ at 100% utilization, respectively [29].

A three-layered tree network topology is considered. The types of ToR, aggregation-layer switches and the core router are HP5920 with $60GBps$ bandwidth (B_{tor}), HP6600 with $180GBps$ bandwidth (B_{agr}), and HP8800 with $430GBps$ bandwidth (B_{cr}) that dissipate $366W$, $405W$ and $3500W$, respectively [14]. For cooling power consumption, we use a time-varying PUE model ranging from 1.25 to 1.55, as in [31].

2) Simulation framework: To simulate a realistic scenario, we utilized the VMs' CPU and memory traces of Bitbrains for a time horizon of two weeks [32]. We used the first week of traces for class generation and exploration phase, and the second week for VMs classification, exploitation phase of ML and for evaluating all the methods. We also used the validated power model of Section III-B to compute DC power consumption by exploiting an in-house simulator tool written in C++, where we coded all the algorithms used in this paper.

The VM allocation and the frequency updating (t_f) are invoked every 1 hour, whereas the cooling system update (t_c) is invoked every 10 minutes. For each experiment, we have considered different number of VMs (from 50 to 1000) in DC.

Data communication between a pair of VMs is modeled by a log-normal distribution [34]. As 80% of the VMs have 800 kB/min traffic among each other while 4% of VMs have 10 times higher traffic [35], we tune log-normal distribution with a mean of 800 and uniform variance in the range of [1,4] for each time slot. The amount of data communication between VMs varies every 5 minutes during the one-hour time slot.

3) Simulation environment: The proposed methods are carried out on a separate server equipped with a 24-core Intel CPU@1.60GHz and 50GB of memory. To solve the ILP, we used the CPLEX 12.3 solver available in GAMS 23.7 [36].

B. Scenarios

1) Scenario I - Optimality assessment: To evaluate the efficiency of the proposed methods we compare them to the ILP-based methods (optimal solutions), for a few number of VMs (50, 100 and 150) and a time horizon of one day. We also take advantage of the fact that in multi-service cloud scenarios, data exchange only occurs between the tenants (VMs) of each service. In other words, we can group together the VMs that exchange data between them, while these groups are isolated and do not share data to other groups. In this scenario we assume a number of groups equal to 20% of the available VMs in the DC and we uniformly distribute the VMs to groups, limiting the number of VMs per group (group size) to 10. We generate different traffic (data communication) between VMs of the same group for each sample during one time slot. We also redistribute the VMs per time slot under the group size limitation. For instance, for the 50 VM scenario we generate 10 groups of sizes between 1-10, and ensuring that each VM is assigned to one group. Moreover, to fairly compare ILP method to other approaches, we compute the total data communication among the servers (total network traffic) for all the methods regardless of network topology constraints.

2) Scenario II - Comparison heuristic, ML and hyper heuristic in large-scale scenarios: Communication patterns contain a wide range of variations from one-to-one to all-to-all traffic between VMs [10]. As opposed to Scenario I, in application-specific private DCs, a high number of VMs communicate with each other, e.g., bank transactions between any two customers. Thus, in this scenario we consider a more general data communication pattern between VMs, assuming that half of the VMs in the DC communicate with each other. The communicating VMs are randomly selected using a uniform distribution. Then, each selected VM is set to exchange data with any other 50% of the VMs, also selected according to a uniform distribution. We analyze the network traffic through different layers as it is considered in the heuristic, ML, and hyper-heuristic methods. Moreover, we increase the number of VMs from 200 to 1000, to compare these methods in a large-scale scenario for a time horizon of one week.

X. Results - Optimality Assessment (Scenario I)

In this section we compare the total energy consumption of DC, QoS (violations caused by overutilized servers), network traffic, number of migrations, and execution time of the algorithms for eight different methods:

- Correlation-aware VM Allocation (CVMA) [7] that is the best in its class to optimize energy consumption.
- Network-aware VM allocation (GH) presented in [10] for network traffic minimization.
- Heuristic: our proposed heuristic (Sect. VI).
- Heuristic-Cap: for a realistic and fair comparison with ILP-based methods, we reduce the servers capacity to 80% during the VM allocation phase to guarantee that no violation occurs due to miss-prediction. This selected cap empirically represents a trade-off between energy efficiency and violation compared to Heuristic. We also assume that all the active servers use the maximum frequency level; i.e. 2.4GHz (100%), to compute violations.
- ML: the proposed ML algorithm (Sect. VII).
- ML-Cap: the proposed ML algorithm with 80% servers capacity cap and setting maximum frequency level.
- ILP-Power: the proposed ILP-based method for DC energy optimization (Sect. V-A).
- ILP-Data: the proposed ILP-based method for data communication optimization (Sect. V-B).

A. Energy efficiency analysis

Figure 5 shows the energy consumption breakdown of the DC including both IT and cooling components. As a result of turning-off more servers, all approaches show an overall energy improvement higher than ILP-Data; on the contrary, ILP-Data further reduces (up to 66%) network energy, as it turns off switches for longer periods of time. Heuristic and ML exhibit less than 2% energy savings compared to CVMA; while providing 10 and 9% improvements compared to GH, respectively. This is because CVMA only considers CPU-load correlation between VMs; but, GH allocates the VMs with high data correlations to fewer servers. On the other hand, ILP-Power only improves up to 5% the results of ML by optimizing the number of active servers. In general, Heuristic-Cap and ML-Cap lead to higher energy consumption, due to the conservative capping approach, that increases the number of used servers.

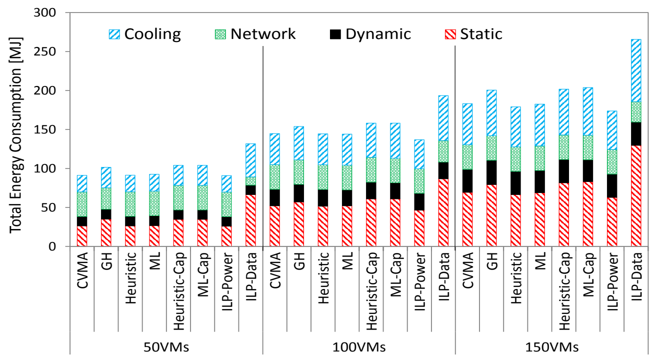

Fig. 5: Energy consumed by the DC for one day.

B. *QoS - Analysis of violations*

Figure 6 (right y-axis) shows the total number of violations, defined as the number of overutilized servers during one day. Heuristic provides a drastic reduction of the violations, from 30 to 87% in worst and best cases compared to ML, respectively. This is because, in the ML approach for a low number of VMs, most of the classes are empty after classification. Therefore, to fill up the servers, ML chooses one VM from the nearest non-empty class, decreasing classification accuracy and leading to violation. On the other hand, Heuristic and ML achieve 94 and 65% improvements compared to CVMA, respectively. GH drastically decreases the number of violations in comparison with the other approaches due to partially filling up the servers. Due to the nature of the ILP-Power, ILP-Data, Heuristic-Cap and ML-Cap, these methods do not present any violation. Thus, they are not shown in Fig. 6.

Figure 6 (left y-axis) shows the average and worst-case amount of overutilized servers for a time horizon of one day, which determines the degree by which the negotiated QoS requirements can be violated. Basically, quality degradation is observed due to the miss-predictions, especially during abrupt workload changes. The results show, for lower number of VMs, Heuristic and CVMA provide better worst-case violation reduction compared to ML. But, for higher number of VMs, the violation of ML remains below the violation of Heuristic and CVMA because classification accuracy is improved. As a result, we obtain 70 and 19% less violations on average and in the worst case for ML compared to Heuristic, and also 88 and 10% improvements compared to CVMA, respectively, for 150 VMs. This figure also shows that GH obtains better results compared to the other approaches due to the servers' underutilizations.

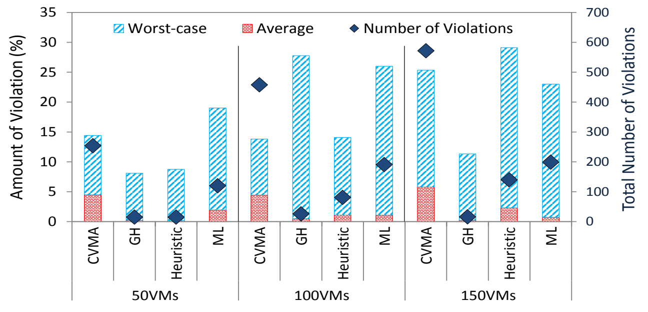

Fig. 6: Average, worst-case percentage amount and total number of violations for one day.

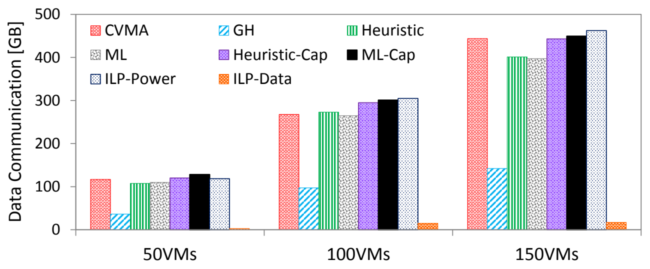

Fig. 7: Total amount of data exchanged among the servers for one day.

C. *Network traffic analysis - data communication*

Figure 7 shows total amount of data exchanged among the servers. Results demonstrate that ILP-Data (optimal solution) reduces the traffic ≈41, 42 and 46x in best case and ≈19, 17 and 21x in the worst case compared to Heuristic, ML and ILP-Power, respectively. This is due to the fact that ILP-Data distributes the non-communicating groups of VMs to different servers to minimize traffic. However, the number of turned-on servers is increased, leading to higher energy consumption. GH achieves up to 3x less network traffic compared to Heuristic and ML since its gaol is to minimize the network traffic. On the contrary, the other approaches first try to use the minimum number of servers and then minimize the data communication.

In this sense, heuristic and ML increase the capability of absorbing time-varying data communication between the servers compared to ILP-Power and CVMA. These results show up to 14, 11 and 3% improvements for ML compared to ILP-Power, CVMA and Heuristic, respectively.

D. *Evaluating the number of migrations*

Table III shows the total number of migrations for one day. CVMA reduces the number of migrations compared to other methods since this considers only CPU-load correlation. On the contrary, ILP-Power and ILP-Data have the highest number of migrations in order to find the optimal allocation solutions. In the best case, ML obtains up to 31 and 25% improvements in the number of migrations compared to GH and Heuristic, respectively, due to the trace classification strategy. Moreover, ML-Cap outperforms Heuristic-Cap by up to 23%.

E. *Execution time of proposed algorithms*

The proposed methods trade-off solution optimality by execution time. To obtain the results shown in Table IV, we run the VM allocation methods for all time slots in 1 day, and we compute its average. The execution time of ILP-based methods is the highest (> 2 hours in some cases), making runtime allocation unfeasible. On the other hand, ML is the fastest

TABLE III: Total number of migrations for one day

Method	50VMs	100VMs	150VMs
CVMA	429	1271	2180
GH	868	2038	3192
Heuristic	715	1862	3003
ML	619	1410	2251
Heuristic-Cap	791	1914	3180
ML-Cap	659	1561	2464
ILP-Power	888	2160	3120
ILP-Data	936	2352	3528

TABLE IV: Execution time (sec.) of the algorithms

Method	50VMs	100VMs	150VMs
CVMA	0.19	0.93	2.49
GH	0.005	0.026	0.073
Heuristics	0.969	3.907	12.897
MLs	0.003	0.005	0.009
ILPs	10.087	287.694	8619.48

algorithm (<10 ms in the worst case) making it particularly suitable to solve large-scale problems.

XI. Results - Large-scale scenario (Scenario II)

In this section we show, for the same metrics than in the previous case, a comparison between the heuristic, ML, hyper-heuristic (Hyper) methods, and the state-of-the-arts.

A. *Energy efficiency analysis*

Figure 8 shows that heuristic, ML and Hyper reach almost similar results for energy consumption ($< 2\%$). Hyper provides better energy savings compared to ML by selecting the Heuristic method in some time slots where Heuristic dramatically outperforms ML in terms of energy consumption. We observe up to 35 and 34% energy improvements for proposed Heuristic and ML algorithms compared to Heuristic-Cap and ML-Cap, respectively, for 800 VMs. For larger scenarios, above 800 VMs, we need to turn-on a new rack and, thus, the second aggregation switch and the core router. Thereby, energy consumption increases due to the higher network energy consumption. Also, Heuristic and ML result in high energy savings compared to GH and CVMA, reducing the number of active servers when the total demand of co-located VMs nearly reaches their server capacity during period.

B. *QoS - Analysis of violations*

Figure 9 (right y-axis) shows that ML provides a violation reduction, up to 15 and 63% compared to Heuristic and CVMA, starting from 400 VMs. On the other hand, for the lower number of VMs, Heuristic performs better than ML. In order to provide a better trade-off, Hyper reduces the number of violations by 11% compared to Heuristic while decreasing the energy consumption compared to ML. Moreover, GH performs better since it is not fully utilizing CPU resources which is not a case for energy efficiency. Also, Heuristic-Cap and ML-Cap reduce the number of violations dramatically because of the cap set on server load.

Figure 9 (left y-axis) shows that Heuristic outperforms ML in terms of the amount of violations for 200 VMs, reaching up to 18% improvement in the worst case. As the number of VMs

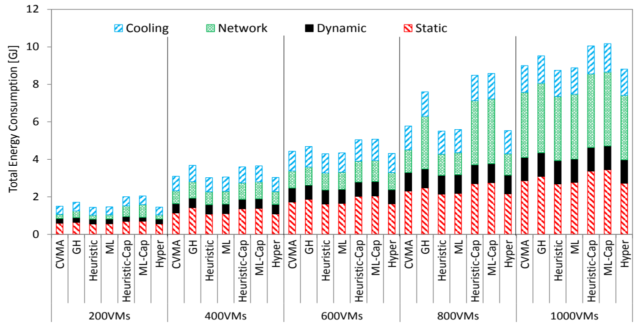

Fig. 8: Energy consumed by DC for one week.

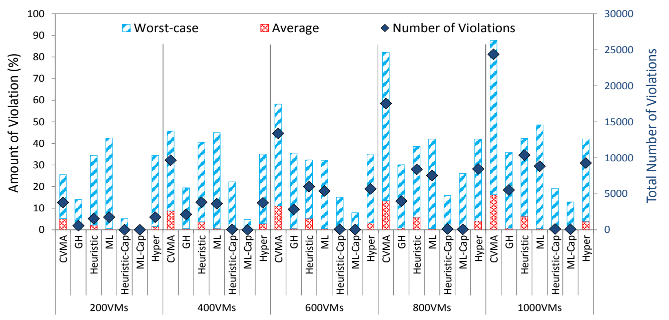

Fig. 9: Average, worst-case percentage amount and total number of violations for one week.

increases, e.g. 800 and 1000 VMs, the violations of ML get closer to Heuristic and GH, inverting the trend for 600 VMs. This is because ML has less control on the worst-case violation during peak loads. Hyper outperforms ML and Heuristic by up to 23 and 14% due to selecting the best method per time slot with lower violation, while leading to only up to 8% overhead compared to both approaches over all cases.

In average, ML provides better results than the other approaches by managing the off-peak VMs load. Finally, for all the cases, Hyper is able to exploit the strengths of Heuristic and ML for providing intermediate solutions.

C. *Multi-layer network traffic analysis*

Figure 10 shows the total traffic through the ToR, aggregation-layer switches, and core router. From 200 to 800 VMs, when the core router is turned off, the results demonstrate that ML reduces the ToRs traffic up to %9 compared to Heuristic; while, Heuristic improves the aggregation-layer up to 4%. Note that for 200 and 800 VMs, traffic in the aggregation and core layers is very low for ML, while for Heuristic they are zero. This increase is due to turning on a server in a new rack. For 1000 VMs, ML provides less ToR and aggregation-layer traffic, but higher core traffic compared to Heuristic. Basically, Heuristic results in lower traffic in the upper layers of the network, due to its fine-tuning capabilities. Comparing the ML to CVMA and GH, we obtain significant improvements especially in upper layers for higher number of VMs, when CVMA and GH are less sensitive to dynamic environments, and their benefits become limited for large problems. By using Hyper, we achieve up to 5 and 7% improvements in ToR and aggregation compared to Heuristic; while 6 and 9% overheads compared to ML, respectively. For the core layer, Hyper improves 53% compared to ML, but presents an overhead of 16% compared to Heuristic.

Following the same trend, Heuristic-Cap outperforms ML-Cap in upper network layers. Differently, we need to turn on the core router for both methods, starting from 800 VMs, due to setting a conservative cap and consequently turning on a server from a new group of racks connected to the aggregation switch.

D. *Evaluating the number of migrations*

Table V represents the total number of migrations for one week. ML reduces the number of migrations by up to 36, 40, and 39% compared to CVMA, GH, and Heuristic, respectively. This happens because in ML, the classification

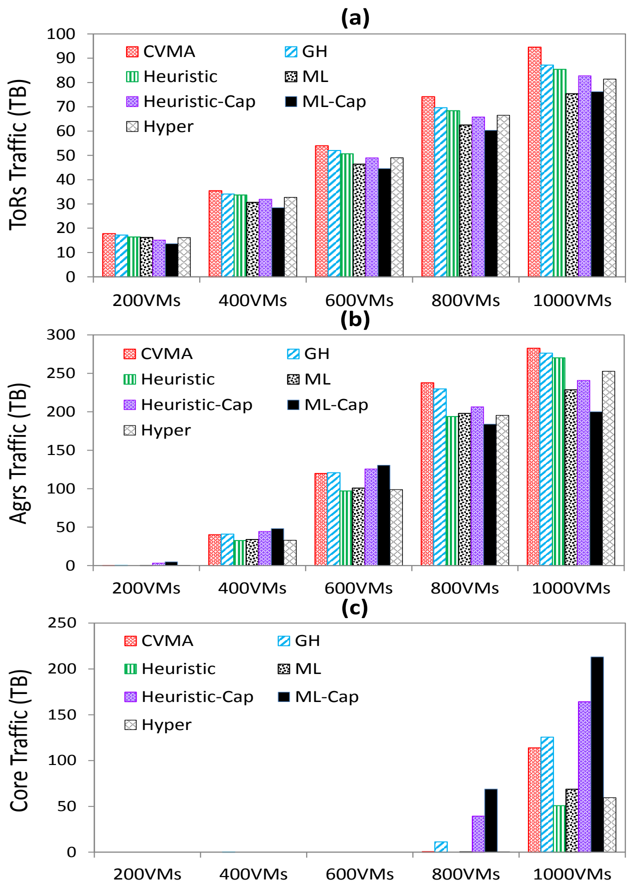

Fig. 10: Network traffic of (a) ToR, (b) Aggregation-layer switches and (c) core router for one week.

accuracy increases for higher number of VMs, thus allowing to place together better candidate VMs. In addition, ML-Cap improves this metric by up to 35% compared to Heuristic-Cap. On the other hand, Hyper achieves 13% improvement and 20% overhead on average over different number of VMs, trading-off the benefits of Heuristic and ML. In the worst case, i.e. 1000 VMs, we only migrate 4.8 and 6.5% of total number of VMs on average every sample for ML and Hyper, respectively, that does not lead to high live migration overhead.

E. Computational overhead (execution time) and discussion

Table VI shows the average execution time of the proposed VM allocation methods for one week of traces. The results follow the same trend than for the small-scale scenario, with Hyper exhibiting a trade-off between the execution time of Heuristic and ML. In summary, the ML method provides almost the same energy efficiency, but dramatically lowers computational overhead when compared to Heuristic, while the heuristic method obtains better network traffic and QoS.

TABLE V: Total number of migrations for one week

Method	200	400	600	800	1000
CVMA	22904	55102	87642	121332	153891
GH	30072	63578	96836	130260	163643
Heuristic	28608	61724	95014	128540	158635
ML	22539	47437	65690	78726	98407
Heuristic-Cap	29406	62607	96061	129465	163007
ML-Cap	23929	49380	69299	86689	105262
Hyper	25609	57471	83143	111935	132822

TABLE VI: Execution time (sec.) of the proposed algorithms under different number of VMs

Methods	200	400	600	800	1000
CVMA	4.45	28.5	88.3	200.6	471.6
GH	1.01	4.12	9.65	17.49	28.6
Heuristics	20.3	88.8	188.7	341.4	519.9
MLs	0.047	0.19	0.421	0.711	1.107
Hyper	10.4	60.86	102.1	199.72	282.35

This is because heuristic methods allow more fine-tuning on the allocation, but present a larger computational overhead, which makes them unsuitable for large-scale scenarios. Our results show that Hyper ensures a good trade-off between solution quality (energy, QoS, and network traffic) using the benefits of both Heuristic and ML approaches.

XII. Conclusion

In this paper we have proposed a two-phase greedy heuristic and a ML method to tackle the challenge of energy- and network-aware VM allocation, evaluating them in terms of energy, network traffic, QoS, migrations and scalability. We have compared them to the optimal solutions (implemented using ILP), and to two algorithms in the state-of-the-art that are the best in their areas. We have presented, for the first time in literature, a novel multi-objective hyper-heuristic method for the VM allocation problem able to find better solutions by leveraging the strengths of heuristic and ML methods, while allowing users to decide on the importance of each metric. Our experimental results have shown that heuristic and ML methods reach similar results in energy consumption ($< 2\%$ difference), consuming only up to 6% more energy than the optimal solution. The ML approach obtains up to 24% server-to-server network traffic improvements when compared to all other methods, and achieving execution time speed-up up to 480x for large-scale problems. On the other hand, the heuristic algorithm results in better QoS and lower traffic in the upper layers of the network structure, due to its fine-tuning capabilities. Finally, the hyper-heuristic algorithm integrates the benefits of heuristic and ML to ensure a good trade-off between solution quality and computational overhead.

References

[1] M. Zapater *et al.*, “Leakage-aware cooling management for improving server energy efficiency,” *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 26, no. 10, pp. 2764–2777, 2015.
[2] L. A. Barroso, J. Clidaras, and U. Hölzle, “The datacenter as a computer: An introduction to the design of warehouse-scale machines, second edition,” *Synth Lect Comput Archit*, vol. 8(3), pp. 1–154, 2013.
[3] X. Meng, V. Pappas, and L. Zhang, “Improving the scalability of data center networks with traffic-aware virtual machine placement,” in *IEEE Conference on Information Communications (INFOCOM)*, 2010, pp. 1154–1162.
[4] E. Pakbaznia and M. Pedram, “Minimizing data center cooling and server power costs,” in *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2009, pp. 145–150.
[5] A. Beloglazov and R. Buyya, “Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints,” *IEEE TPDS*, vol. 24, no. 7, pp. 1366–1379, 2013.
[6] A. Pahlevan, P. G. d. Valle, and D. Atienza, “Exploiting cpu-load and data correlations in multi-objective vm placement for geo-distributed data centers,” in *Design, Automation Test in Europe Conf. (DATE)*, 2016, pp. 1333–1338.
[7] J. Kim, M. Ruggiero, D. Atienza, and M. Lederberger, “Correlation-aware virtual machine allocation for energy-efficient datacenters,” in *DATE*, 2013, pp. 1345–1350.

[8] L. Chen and H. Shen, "Consolidating complementary VMs with spatial/temporal-awareness in cloud datacenters," in *IEEE Conference on Computer Communications*, 2014, pp. 1033–1041.

[9] "The future of data center wide-area networking," in *Forrester Research*, 2010.

[10] O. Biran *et al.*, "A stable network-aware VM placement for cloud systems," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2012, pp. 498–506.

[11] Z. A. Mann, "Allocation of virtual machines in cloud data centers–a survey of problem models and optimization algorithms," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 11:1–11:34, 2015.

[12] A. Iranfar, S. N. Shahsavani, M. Kamal, and A. Afzali-Kusha, "A heuristic machine learning-based algorithm for power and thermal management of heterogeneous mpsocs," in *ISLPED*, 2015, pp. 291–296.

[13] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *International Conference on Practice and Theory of Automated Timetabling III*, 2001, pp. 176–190.

[14] S. Esfandiarpoor, A. Pahlavan, and M. Goudarzi, "Structure-aware online virtual machine consolidation for datacenter energy improvement in cloud computing," *Computers & Electrical Eng.*, pp. 74–89, 2015.

[15] D. Meisner *et al.*, "Power management of online data-intensive services," in *ACM Int. Symp. on Comput. Archit. (ISCA)*, 2011, pp. 319–330.

[16] E. Ahvar *et al.*, "Cacev: A cost and carbon emission-efficient virtual machine placement method for green distributed clouds," in *IEEE International Conference on Services Computing*, 2016, pp. 275–282.

[17] A. Verma *et al.*, "Server workload analysis for power minimization using consolidation," in *USENIX Annual Tech. Conf.*, 2009, pp. 28–28.

[18] X. Meng *et al.*, "Efficient resource provisioning in compute clouds via VM multiplexing," in *ACM International Conference on Autonomic Computing (ICAC)*, 2010, pp. 11–20.

[19] W. Lin *et al.*, "Design and theoretical analysis of virtual machine placement algorithm based on peak workload characteristics," *Soft Comput.*, vol. 21, no. 5, pp. 1301–1314, 2017.

[20] X. Ruan and H. Chen, "Performance-to-power ratio aware virtual machine (VM) allocation in energy-efficient clouds," in *IEEE International Conference on Cluster Computing*, 2015, pp. 264–273.

[21] J. V. Wang, K. Y. Fok, C. T. Cheng, and C. K. Tse, "A stable matching-based virtual machine allocation mechanism for cloud data centers," in *IEEE World Congress on Services (SERVICES)*, 2016, pp. 103–106.

[22] F. Farahnakian *et al.*, "Multi-agent based architecture for dynamic VM consolidation in cloud data centers," in *EUROMICRO Conference on Software Engineering and Advanced Applications*, 2014, pp. 111–118.

[23] S. S. Masoumzadeh and H. Hlavacs, "A cooperative multi agent learning approach to manage physical host nodes for dynamic consolidation of virtual machines," in *IEEE Symposium on Network Cloud Computing and Applications (NCCA)*, 2015, pp. 43–50.

[24] ——, "Integrating vm selection criteria in distributed dynamic VM consolidation using fuzzy q-learning," in *International Conference on Network and Service Management (CNSM)*, 2013, pp. 332–338.

[25] V. Ravi and H. S. Hamead, "Reinforcement learning based service provisioning for a greener cloud," in *Int. Conf. on Eco-friendly Computing and Communication Systems (ICECCS)*, 2014, pp. 85–90.

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

[27] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *IEEE Int. Conf. on Advanced Information Networking and Applications (AINA)*, 2010, pp. 400–407.

[28] A. Pahlavan, M. Momtazpour, and M. Goudarzi, "Power reduction in hpc data centers: a joint server placement and chassis consolidation approach," *Journal of Supercomputing*, vol. 70, pp. 845–879, 2014.

[29] J. C. Salinas-Hilburg *et al.*, "Unsupervised power modeling of co-allocated workloads for energy efficiency in data centers," in *DATE*, 2016, pp. 1345–1350.

[30] D. Kusic *et al.*, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009.

[31] J. Kim, M. Ruggiero, and D. Atienza, "Free cooling-aware dynamic power management for green datacenters," in *Int. Conf. on High Performance Computing and Simulation (HPCS)*, 2012, pp. 140–146.

[32] S. Shen, V. v. Beek, and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters," in *CCGrid*, 2015, pp. 465–474.

[33] I. Mesecan and I. Ö. Bucak, "Searching the effects of image scaling for underground object detection using kmeans and knn," in *European Modelling Symposium (EMS)*, 2014, pp. 180–184.

[34] D. S. Dias and L. H. M. K. Costa, "Online traffic-aware virtual machine placement in data center networks," in *Global Information Infrastructure and Networking Symposium (GIIS)*, 2012, pp. 1–8.

[35] M. H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya, "Network-aware virtual machine placement and migration in cloud data centers," *Bagchi S (ed) Emerging research in cloud distributed computing systems, Chap 2. Information Science Reference, Hershey PA*, pp. 42–91, 2015.

[36] R. E. Rosenthal, "Gams-a user's guide," in *GAMS Development Corporation*, 2016.

Ali Pahlevan is currently a Ph.D. candidate in Electrical Engineering (EE) in the Embedded Systems Laboratory (ESL) at Swiss Federal Institute of Technology Lausanne (EPFL). He received his M.Sc. degree in Computer Engineering from Sharif University of Technology in 2012, and B.Sc. degree in Computer Engineering from Ferdowsi University of Mashhad in 2010. His research interests focus on system-level energy optimization techniques in the area of data centers and cloud computing.

Xiaoyu Qu received both her M.Sc. and B.Sc. degrees in Control Science and Engineering from Beijing Institute of Technology in China. During M.Sc. degree, she has been an exchanged student in Electronic Industry Center Lab (CEI) at Universidad Politecnica de Madrid. After graduation, in 2016, she has been an intern in ESL at EPFL. Her research interests include energy efficiency in data centers.

Marina Zapater is currently is a Post-Doctoral researcher in the ESL at EPFL. She was Visiting Assistant Professor in the Computer Architecture Department at Universidad Complutense de Madrid, Spain, in the academic year 2015-2016. She received her Ph.D. degree in Electronic Engineering from Universidad Politecnica de Madrid in 2015, an M.Sc. in Telecommunication Engineering degree and a M.Sc. in Electronic Engineering degree, both from the Universitat Politecnica de Catalunya, in 2010. Her research interests include proactive and reactive thermal and power optimization of complex heterogeneous systems, energy efficiency in data centers, ultra-low power architectures and embedded systems. In this area, she has co-authored over 25 publications in top-notch international conferences and journals, and she has participated in several national and international research projects. She is a member of IEEE and CEDA and has served as TPC member of several conferences, including VLSI-SoC and MCSoC.

David Atienza (M'05-SM'13-F'16) is associate professor of electrical and computer engineering, and director of the Embedded Systems Laboratory (ESL) at the Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland. His research interests include system-level design methodologies for high-performance multi-processor system-on-chip (MP-SoC) and low-power embedded systems, including new 2-D/3-D thermal-aware design for MPSoCs and many-core servers, ultra-low power system architectures for wireless body sensor nodes, HW/SW reconfigurable systems, dynamic memory optimizations, and network-on-chip design. He is a co-author of more than 250 publications in peer-reviewed international journals and conferences, several book chapters, and seven U.S. patents in these fields. He has earned several best paper awards and he is (or has been) an Associate Editor of IEEE TC, IEEE D&T, IEEE T-TCAD, IEEE T-SUSC and Elsevier *Integration*. He was the Technical Programme Chair of IEEE/ACM DATE 2015 and General Programme Chair of IEEE/ACM DATE 2017. Dr. Atienza received an ERC Consolidator Grant in 2016, the IEEE CEDA Early Career Award in 2013, the ACM SIGDA Outstanding New Faculty Award in 2012, a Faculty Award from Sun Labs at Oracle in 2011, and was Distinguished Lecturer (period 2014-2015) of IEEE CASS. He is an IEEE Fellow and Senior Member of ACM.