An Approach to Balance Maintenance Costs and Electricity Consumption in Cloud Data Centers

Luca Chiaraviglio, *Senior Member, IEEE*, Fabio D'Andreagiovanni, *Member, IEEE*, Riccardo Lancellotti, *Member, IEEE*, Mohammad Shojafar, *Member, IEEE*, Nicola Blefari-Melazzi, *Senior Member, IEEE*, and Claudia Canali, *Member, IEEE*

Abstract—We target the problem of managing the power states of the servers in a Cloud Data Center (CDC) to jointly minimize the electricity consumption and the maintenance costs derived from the variation of power (and consequently of temperature) on the servers' CPU. More in detail, we consider a set of virtual machines (VMs) and their requirements in terms of CPU and memory across a set of Time Slot (TSs). We then model the consumed electricity by taking into account the VMs processing costs on the servers, the costs for transferring data between the VMs, and the costs for migrating the VMs across the servers. In addition, we employ a material-based fatigue model to compute the maintenance costs needed to repair the CPU, as a consequence of the variation over time of the server power states. After detailing the problem formulation, we design an original algorithm, called Maintenance and Electricity Costs Data Center (MECDC), to solve it. Our results, obtained over several scenarios from a real CDC, show that MECDC largely outperforms two reference algorithms, which instead either target the load balancing or the energy consumption of the servers.

Index Terms—Cloud computing, cloud data center, maintenance costs, electricity costs, fatigue, energy-efficiency

1 Introduction

Data Centers (DCs) have become a key aspect of the Information and Communication Technology (ICT) sector. Historically, the idea of exploiting DCs for computing tasks dates back to the first half of the 19th century, when different prominent researchers defined the concept of global brain [1], [2], with the goal of providing encyclopaedic ways of knowledge. Since then, the incredible growth in the ICT sector, including the improvements in HardWare (HW) manufacturing, as well as the almost-infinite features provided by SoftWare (SW), have completely revolutionized the possibility of exploiting DCs for computing purposes. Nowadays, DCs are widely spread worldwide to sustain a variety of applications, such as web browsing, streaming, high definition videos, and cloud storage. Not surprisingly, DCs generally adopt the cloud computing paradigm [3], [4], according to which the virtualized applications (and entire operating systems) run over a set of distributed physical servers, which may be even located in different continents. Hence, the management of a Cloud Data Center (CDC) is an aspect of fundamental importance for the DC owner (which is referred as a *content provider* from here on).

In an era where the amount of computing information is constantly growing [5], a primary need for a content provider is to efficiently manage CDCs. Apart from the fixed costs, which are related to the installation of CDCs equipment [6], a big worry for a content provider is how to deal with the CDCs power consumption and the related electricity costs [7]. In this context, the content provider has to face the large amount of power consumed by its own CDCs. As a result, the decrease of power consumption in CDCs has been traditionally a hot topic [8]. In line with this trend, different works (see e.g., [9], [10]) target the reduction of power for the servers in a CDC through the management of their power states. Among them, the application of a Sleep Mode (SM) state to a subset of servers is a very promising approach in order to save energy [11], [12]. More in detail, thanks to the fact that the traffic from users is not constant and generally varies across the different hours of the day, it is possible in a CDC to put different servers in SM, and to concentrate the users traffic on a subset of servers, which remain in an Active Mode (AM). In this way, a reduction of power and, consequently, a reduction of the associated electricity costs paid by the content provider are achieved.

Although the application of SM is able to ensure lower electricity costs compared to the case in which all the servers are always powered on, the transitions between SM and AM, especially when they are applied over periods of several months and years, tend to have a negative effect on the maintenance costs paid by the content provider [13]. More in detail, when the server is put in SM, a prompt decrease in the temperature of its components (especially for CPU and

- *L. Chiaraviglio and N. Blefari-Melazzi are with the Department of Electronic Engineering, University of Rome Tor Vergata, Roma, 00133, Italy, and CNIT, Parma 43100, Italy. E-mail: {luca.chiaraviglio, nicola.blefari-melazzi}@uniroma2.it.*
- *F. D'Andreagiovanni is with CNRS (France) and the Laboratory "Heudiasyc" (CNRS UMR 7253), Sorbonne Universités, Université de Technologie de Compiègne, France. E-mail: d.andreagiovanni@hds.utc.fr.*
- *R. Lancellotti and C. Canali are with the Department of Engineering Enzo Ferrari, University of Modena and Reggio Emilia, Modena, MO 41121, Italy. E-mail: {riccardo.lancellotti, claudia.canali}@unimore.it.*
- *M. Shojafar is with the Department of Mathematics, University of Padua, Padova 35122, Italy. E-mail: mohammad.shojafar@math.unipd.it.*

Manuscript received 13 Dec. 2017; revised 27 Apr. 2018; accepted 15 May 2018. Date of publication 18 May 2018; date of current version 6 Dec. 2018. (Corresponding author: Luca Chiaraviglio.)
Recommended for acceptance by S. Guo.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSUSC.2018.2838338

memories) is observed [14]. Specifically, the temperature drops from pretty high values (typically higher than 70-80 degree [Celsius]) to the room temperature, which is typically cooled and kept around 20 degree [Celsius]. On the other hand, the opposite effect on the temperature is observed when the server passes from SM to AM. The variation of temperature on the electronics components, especially when it is repeated over time, tends to introduce thermal fatigue effects [15], [16]. This phenomenon is similar to the mechanical fatigue experienced by an airplane fuselage, subject to cabin pressurization and depressurization over different flights, which may deteriorate it in the long term [17]. In a similar way, the HW equipment, when it is subject to large temperature transitions, tends to increase its failure rate. More in detail, fatigue (and crack) effects are experienced, for example, by the solder joints connecting the CPU/memories to the motherboard [18]. As a consequence, a server subject to frequent AM/SM transitions will experience failure events more often, compared to the case in which it is always left in AM, thus increasing the associated maintenance costs in order to fix and/or replace the failed components. In the worst case, the maintenance costs will be even larger than the electricity saved from the application of SM, thus producing a monetary loss to the content provider [13].

This context poses several challenges: What is the impact of the maintenance costs on the total costs? Is it beneficial to leverage the tradeoff between electricity consumption and maintenance costs? How to optimally formulate the problem? How to design an efficient algorithm to tackle it? The goal of this paper is to shed light on these issues. More in detail, we first present a simple (yet effective) model to compute the maintenance costs, given the variation over time of the power states for a set of servers. In addition, we adopt a detailed model to compute the power consumed by the CDC. Specifically, our power model takes into account the CPU-related electricity costs of the servers, the costs for transferring data among the servers, and the costs for migrating the Virtual Machines (VMs) running on the servers. After formulating the problem of jointly reducing the CDC electricity consumption and the related maintenance costs, we propose a new algorithm, called Maintenance Energy Costs Data Center (MECDC), to tackle it. Our results, obtained over several scenarios from a real CDC, clearly show that our solution is able to wisely leverage the tradeoff between maintenance and electricity costs in order to provide monetary savings for the content provider. On the other hand, we show that other strategies, either targeting the VMs load balancing, or the servers energy consumption, tend to notably increase the total costs. To the best of our knowledge, none of the previous works in the CDC research field has conducted a similar analysis.

Although the results reported in this paper are promising, we point out that other costs than the ones considered here may increase the maintenance bill. Specifically, the cost of regular updates, due to HW/SW upgrades, may have an impact on the maintenance costs paid by the content provider. In addition, the adoption of renewable energy sources may also vary the electricity bill. Both these issues, which are not considered in this work, can be potentially added in our framework.

The rest of the paper is organized as follows. Related works are reviewed in Section 2. The reference CDC architecture is briefly overviewed in Section 3. Section 4 presents the considered models to compute the maintenance costs and the electricity costs in a CDC. The problem of jointly managing the electricity and the maintenance costs triggered by fatigue processes is formulated in Section 5. Section 6 details the MECDC algorithm. The considered scenarios and the setting of the input parameters are detailed in Section 7. Results are reported in Section 8. Finally, Section 9 concludes our work.

2 Related Work

In the following, we briefly discuss the main literature in CDC related to our work. We first describe solutions targeting the management of energy and/or electricity in CDCs. Then, we move our attention to researches targeting the management of CDC failures.

2.1 Energy and Electricity Management in CDCs

Features such as electricity, power, as well as computing and network management tasks are addressed in [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29]. A detailed overview of energy consumption models in Data Center (DC) is provided by Dayarathna et al. in [19]. In this context, several works target the management of a CDC by: i) providing algorithms for VM live migrations [20], [21], [22], ii) considering distributed server/CDC applications [23], [24], [25], [26], iii) focusing on business process management [27], and iv) detailing memory and storage management solutions [28], [29].

Focusing on the aspect of VM live migrations, Voorsluys et al. [20] adopt live migration of VMs, with the goal of reducing energy in the CDC while guaranteeing the performance to applications. However, this work does not consider the server maintenance costs. Moreover, the costs of VM migration and data transferring between VMs in a CDC environment are not taken into account. Liu et al. in [22] present a cost-aware learned knowledge method and an adaptive network bandwidth management, by applying VM live migration estimation to achieve power saving in the CDC. Soni et al. in [23] derive computing cost models for the CDC such that they try to cover the VMs' over/under loadings based on priority and states. Indeed, their proposed algorithm is able to manage load distribution among various applications running in each VM. Bi et al. in [25] present a queue-aware multi-tier application model inside the CDC. In addition, they compute the number of servers that must be allotted to each tier in order to meet the response time per application per server. They also consider the CPU resources per-VM in the CDC. However, a live VM migration is not performed. Finally, Han et al. in [26] present an adaptive cost-aware elasticity method in order to scale up/down multi-tier cloud applications to meet run-time varying application demands. Nevertheless, the complexity of the proposed model in computational management is quadratic per-application. Focusing on the memory and storage management, Song et al. in [29] employ power performance information to estimate the desired storage and memory parameters in order to preserve energy and costs in the CDC. It is important to note that their quasi-analytical performance modeling can be accurate, but it requires a deep understanding of each individual application running on the VM and the server. Therefore, a consistent amount of preliminary information is needed and, as a consequence, the pre-processing time of the problem may sensibly increase.

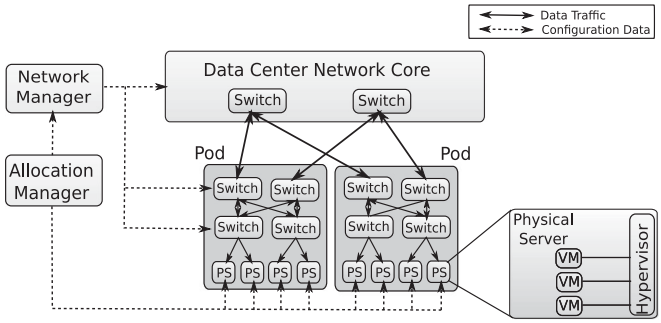


Fig. 1. Cloud data center architecture.

2.2 Failure Management in CDCs

Server failure is recognized as an important cost component for the cloud, see e.g., Greenberg et al. [30]. Therefore, different works target the reduction of the impact of the failure events by proposing efficient DC architectures. In particular, Guo et al. propose Dcell [31], a scalable and recursive architecture which is also fault-tolerant. Greenberg et al. [32] present VL2, a scalable and flexible DC network which is tolerant to failures experienced by networking equipment. Guo et al. [33] details BCube, an architecture for modular DCs, which is able to guarantee a graceful performance degradation as the server failure rate increases. Moreover, according to Kliazovich et al. [34], when the DC temperatures are not kept within their operational limits the HW reliability is decreased, thus bringing to a potential violation of Service Level Agreements (SLAs). In addition, the optimization of thermal states and cooling system operation is recognized as a challenge by Beloglazov et al. [10]. A detailed analysis of failures in a DC is performed by Gill et al. [35]. However, the work is mainly focused on network devices and not on servers like in our case. Eventually, a characterization of the HW components of the servers in terms of reliability is performed by Vishwanath et al. [36]. In particular, this work reports that the failure in one of the server HW components is a common event experienced in large DCs. In [37] Zhang et al. advocate the need of taking availability into consideration while mapping VMs. In this context, Fan et al. [38] explore the problem of mapping service function chains with guaranteed availability. Finally, Jhawar and Piuri [39] propose an approach to measure the effectiveness of fault tolerance mechanisms in Infrastructure as a Service (IaaS) cloud, by also providing a solution to select the best mechanism satisfying the users requirements.

3 Cloud Data Center Architecture

Fig. 1 reports the main building blocks of the considered CDC architecture. More in detail, the CDC is composed of VMs, hypervisors, Physical Servers (PSs), switches and management entities. Each VM is hosted in a PS. The set of VMs in a PS is managed by an hypervisor. Moreover, the PSs are grouped in Pods. The interconnection between PSs in the same Pod is realized by means of a redundant set of switches and physical links. In addition, a DC network, again composed of switches and physical links, provides connectivity between the different Pods. Moreover, a centralized network manager (top left part of the figure) is then in charge of managing the set of networking devices, e.g., by providing software-defined functionalities. Finally, an allocation manager (mid left part of the figure) distributes the VMs over the PSs, by ensuring that each VM receives the required amount of CPU and memory from the PS hypervisor.

Focusing on the tasks performed by the allocation manager, this element is in charge of running the proposed VMs' allocation algorithm, which is able to leverage the tradeoff between electricity costs and maintenance costs by acting on the PSs power states. In our work, we assume that time is discretized in Time Slots (TSs), and that the allocation algorithm is run for every TS. Given: i) a current TS τ and the corresponding VMs requests in terms of CPU and memory;[1] ii) the power state of the PSs (AM or SM) and the allocation of VMs at the previous TS $\tau - 1$; the allocation manager computes the allocation of VMs for TS τ. Eventually, the allocation manager notifies the PSs that need to be put in AM/SM for the current TS. In case a PS was in AM at previous TS and needs to be put in SM at current TS, the allocation manager interacts with the PS operating system to gracefully halt the machine.

4 Costs Models

We first consider the computation of the maintenance and electricity costs for a generic TS t, whose duration is denoted by $\delta(t)$ [h]. We initially present the model to compute the maintenance costs in a CDC subject to fatigue effects. We then detail the model adopted to compute the electricity costs. Finally, we discuss the interdependence between the two models.

4.1 Maintenance Cost Model

We first introduce a failure model in order to take into account the impact of power transitions on the PS. We start from [13], in which authors present a generic model that can be applied to computing equipment. In particular, the proposed model is representative of failures involving the CPU, which is one of the most critical (and hot) components in a PS.[2] We denote by S the set of PSs and we focus on a generic $s \in S$ in the CDC. The total Failure Rate (FR) $\phi_s^{TOT}(t)$ for PS s at TS t is defined as:

$$\phi_s^{TOT}(t) \triangleq \phi_s^{AM}\left(1 - \frac{\tau_s^{SM}(t)}{\tau^{ALL}(t)}\right) + \phi_s^{SM} \cdot \frac{\tau_s^{SM}(t)}{\tau^{ALL}(t)} + \frac{\eta_s}{N_s^F}, \quad (1)$$

where ϕ_s^{AM} [1/h] is the Failure Rate (FR) of the PS when it is always kept in AM (i.e., no SM is applied), $\tau_s^{SM}(t)$ [h] is the amount of time the PS has spent in SM (from the beginning of the simulation up to current TS t), $\tau^{ALL}(t)$ [h] is the total amount of time under consideration, ϕ_s^{SM} [1/h] is the PS FR when it is always left in SM (i.e., no AM is applied), η_s [1/h] is the frequency of power state transitions between SM and AM, and N_s^F is the number of AM-SM cycles before a failure occurs. As reported in [13], the main assumptions of this model are that the failures are assumed to be statistically independent of each other and that their effect is additive. By observing in more detail Eq. (1), we can notice two different effects. Specifically, when the amount of time in SM $\tau_s^{SM}(t)$ is increased, the resulting FR $\phi_s^{TOT}(t)$ tends to the value ϕ_s^{SM}, which is, in general, lower than ϕ_s^{AM} (thanks to

1. In this way, the VM resources are expressed in terms of CPU and memory requirements for the current TS. Clearly, the current TS is the same across all the requests.

2. The extension of the model to other components, and their mutual interactions, is left for future work.

the fact that the temperature in SM is much lower compared to the AM case). On the other hand, the number of transitions between AM and SM tends to increase with time, thus increasing the last term of Eq. (1), and consequently the total FR $\phi_s^{TOT}(t)$. This last term tends to dominate the FR, especially when the amount of time under consideration $\tau^{ALL}(t)$ is in the order of months/years.

Using the elements introduced above, we compute the maintenance costs C_M^{TOT} [$] at TS t for all the PSs in the CDC as:

$$C_M^{TOT}(t) = K_R \cdot \delta(t) \cdot \sum_{s \in S} \phi_s^{TOT}(t), \quad [\$], \tag{2}$$

where K_R [$] is the reparation cost for one PS (i.e., the cost for fixing the PS without the need to replace it with a new one), and $\delta(t)$ is the duration of the considered TS. In this work, we assume that the PS failures can be repaired by, e.g., the substitution of only the failed components with new ones. We believe that this assumption is more realistic compared to the case in which a PS is always replaced with a new one each time a failure is experienced. Finally, we stress the fact that the total maintenance costs $C_M^{TOT}(t)$ may include also the costs for HW upgrades and SW updates, as well as scheduled maintenance operations. These terms can be added as additional costs in Eq. (2), and they are left for future work.

In the following, we introduce a simple metric, called Acceleration Factor (AF), to better capture the model features. More in detail, the AF, which is a metric commonly adopted in material fatigue researches [16], [40], is defined as the ratio between the observed FR $\phi_s^{TOT}(t)$ and the FR by keeping the PS always in AM, i.e., ϕ_s^{AM}. More formally, we have:

$$AF_s^{TOT}(t) \triangleq \frac{\phi_s^{TOT}(t)}{\phi_s^{AM}} = 1 - \left(1 - AF_s^{SM}\right) \frac{\tau_s^{SM}(t)}{\tau^{ALL}(t)} + \Psi_s \cdot \rho_s(t), \tag{3}$$

where AF_s^{SM} is defined as $\frac{\phi_s^{SM}}{\phi_s^{AM}}$ (which is typically lower than 1 as the FR in SM is lower than the one in AM), $\rho_s(t)$ is the total number of power state transitions up to the current TS t and Ψ_s is a weight parameter. Consequently, we express the total failure rate in Eq. (2) as $\phi_s^{TOT}(t) = AF_s^{TOT}(t) \cdot \phi_s^{AM}$.

When $AF_s^{TOT}(t) < 1$, the PS lifetime (i.e., the time between two failure events) is higher compared to the case in which the PS s is always left in AM. On the other hand, when $AF_s^{TOT}(t) > 1$, the lifetime is lower compared to the AM case. The value of $AF_s^{TOT}(t)$ gives exactly the amount of lifetime reduction for the PS, e.g., if $AF_s^{TOT}(t) = 30$, the PS will experience a lifetime reduction of 30 times compared to the case in which it is always kept in AM. Clearly, the application of different power states has an impact on the values of $AF_s^{TOT}(t)$. More in detail, when the observation period (i.e., the time passed from the beginning of the experiment up to the current time slot) is in the order of months/years, the term $\Psi_s \cdot \rho_s(t)$ becomes predominant, i.e., the application of different power states tends to increase $\rho_s(t)$, and consequently the AF. Finally, we can note that the AF is influenced by the parameters $\tau_s^{SM}(t)$ and $\rho_s(t)$, which depend on the specific policy used to put the PS in SM/AM, and by parameters AF_s^{SM} and Ψ_s, which instead depend on the materials used to build the CPU (and their strength against fatigue effects). In principle, CPUs exhibiting higher values of Ψ_s are more prone to fatigue effects, and consequently to lifetime degradation. The actual setting of parameters AF_s^{SM} and Ψ_s will be discussed in more detail in Section 7.

4.2 Electricity Cost Model

We model the electricity costs as the sum of three different contributions: i) the data processing costs on the PSs, ii) the data transferring costs among the VMs located on different PSs, and iii) the costs for migrating the VMs across different PSs. The following sections detail the different cost components.

4.2.1 Data Processing Costs

We adopt the assumption of [10], according to which the power consumption of each PS in AM is proportional to the CPU utilization due to data processing tasks running on the hosted VMs. On the other hand, when the PS is in SM, we assume that its power consumption is negligible. We denote the total electricity costs due to processing tasks at TS t as $C_E^{PROC}(t)$. More formally, we have:

$$C_E^{PROC}(t) = K_E \cdot \delta(t) \sum_{s \in S} \big[u_s(t)\big(P_s^{MAX} - P_s^{IDLE}\big) + O_s(t) \cdot P_s^{IDLE}\big], \quad [\$] \tag{4}$$

where K_E [$/Wh] is the hourly electricity cost, $\delta(t)$ [h] is the TS duration, $u_s(t)$ is the CPU utilization of the PS s at current TS (ranging between 0 and 1), P_s^{MAX} [W] is the power consumption of s when its CPU is fully utilized, P_s^{IDLE} [W] is the power consumption of s when its CPU is idle, and $O_s(t)$ is the power state of s at TS t (0 if it is in SM, 1 otherwise). Note that, when the PS is in SM (i.e., $O_s(t) = 0$), it holds that $u_s(t) = 0$.

4.2.2 Data Transferring Costs

We then consider the electricity costs derived from the exchange of data between VMs running on different PSs. As common in literature (see e.g., [41], [42]), we assume that the total costs due to data transferring are the sum of a static term, which considers the power consumed by the network interfaces of the PS, and a linear one, which instead takes into account the amount of data transferred between VMs. The total costs due to data transferring, which are denoted with $C_E^{TR}(t)$, are then expressed as:

$$C_E^{TR}(t) = K_E \cdot \delta(t) \sum_{s \in S} \Big[O_s(t) \cdot P_s^{TR-IF} + \sum_{\substack{\sigma \in S \\ \sigma \neq s}} \sum_{m,n \in M} d_{mn}^{s\sigma}(t) \cdot P_{s\sigma}^{TR-NET}\Big], \quad [\$] \tag{5}$$

where M is the set of VMs in the CDC, P_s^{TR-IF} [W] is the power of the network interfaces of PS s, $d_{mn}^{s\sigma}(t)$ [Mb] is the amount of data traffic exchanged during TS t between VM m on PS s and VM n on PS σ (which is equal to 0 if either PS s or PS σ is in SM), and $P_{s\sigma}^{TR-NET}$ [W/Mb] is the power consumption consumed for transferring one [Mb] of information between PS s and PS σ (by assuming that VM m is hosted in PS s, and that VM n is located in PS σ).

4.2.3 Migration Costs

Finally, we consider the costs that are paid when the VMs are moved across the PSs. For example, a typical event requiring VM migration is the activation of SM on a PS.

Before the PS applies SM, all the VMs running on it have to be moved to other PS(s). We assume that the VM migration involves the whole copy of the VM memory from the old PS to the new one.[3] Eventually, the process of copying the memory requires an additional amount of overhead power, which needs to be properly taken into account. This amount of power is driven by the fact that VM migration introduces a performance degradation, which may be even in the order of 10 percent according to [43]. The migration costs at TS t, which are denoted with $C_E^{MIG}(t)$, are then defined as:

$$C_E^{MIG}(t) = K_E \cdot \sum_{s,\sigma \in S} \sum_{m \in M} y_{s\sigma m}(t) \big[\mu_m(t) \cdot P_{s\sigma}^{TR-NET} + P_s^{OH} + P_\sigma^{OH}\big], \quad [\$], \tag{6}$$

where $y_{s\sigma m}(t)$ is a binary variable taking value 1 if VM m on PS s is migrated to PS σ at TS t (0 otherwise), $\mu_m(t)$ [Mb] is the amount of memory consumed by the VM m during TS t, $P_{s\sigma}^{TR-NET}$ [W/Mb] is again the power consumption consumed for transferring one [Mb] of information between PS s and PS σ, P_s^{OH} [W] and P_σ^{OH} [W] are the amount of overhead power consumed during the migration process by PS s and σ, respectively.

4.2.4 Total Electricity Costs

The electricity costs consumed at TS t by the CDC are then computed as the sum of the considered costs:

$$C_E^{TOT}(t) = \big[C_E^{PROC}(t) + C_E^{TR}(t) + C_E^{MIG}(t)\big], \quad [\$] \tag{7}$$

4.3 Interdependence between the Costs Models

The presented electricity and maintenance costs models are strictly independent of each other. Let us consider for simplicity the case in which a generic PS s was in AM at previous TS and it is put in SM at current TS. In this case, the number of power state transitions $\rho_s(t)$ is increased. This inevitably increases the AF of the sth PS reported in Eq. (3), and consequently the reparation costs in Eq. (2). On the other hand, by imposing the SM state, $O_s(t)$ is set to 0. Therefore, the data processing costs in Eq. (4) and the data transferring costs in Eq. (5) are equal to 0 for PS s. On the other hand, the VMs running on the PS will be moved to other PSs, thus increasing the migration costs in Eq. (6). In a similar way, the power state change from SM in the previous TS to AM in the current time slot also tends to increase the reparation costs, while also increasing the electricity costs.

In this context, a natural question is: How to set the power states for the whole set of PSs in the CDC in order to leverage the tradeoff between the costs? To answer this question, we optimally formulate in the next section the problem of minimizing the total costs in a CDC over a set of TSs.

5 Problem Formulation

We first consider the extension of our cost model by introducing the set of TSs, which is denoted by $\mathcal{T}$. Then, we target the problem of jointly managing maintenance and electricity costs in the CDC over the whole set of TSs. We initially detail each set of constraints, and then we provide the entire formulation.

5.1 Maintenance Costs Constraints

We first consider the constraints related to the computation of the maintenance costs. We initially introduce the variable $\tau^{ALL}(t)$ [h] to compute the total amount of time elapsed from the initial TS up to TS $t \in \mathcal{T}$. $\tau^{ALL}(t)$ is computed as:

$$\tau^{ALL}(t) = \tau^{ALL}(t-1) + \delta(t), \quad \forall t \in \mathcal{T}, \tag{8}$$

where $\tau^{ALL}(t-1)$ [h] is the total elapsed time up to TS $(t-1)$ and $\delta(t)$ [h] is the duration of current TS t.

We then denote with $\tau_s^{SM}(t)$ [h] the total time in SM for PS s up to TS t. $\tau_s^{SM}(t)$ is then computed as:

$$\tau_s^{SM}(t) = \tau_s^{SM}(t-1) + \delta(t)[1 - O_s(t)], \quad \forall s \in S, \forall t \in \mathcal{T}, \tag{9}$$

where $\tau_s^{SM}(t-1)$ [h] is the total time in SM for PS s up to TS $(t-1)$, and $O_s(t)$ [units] is a binary variable for the power state of PS s, taking value 1 if PS s is in AM at TS t, 0 otherwise.

We then introduce the binary variable $z_s(t)$ [units], which takes value 1 if PS s has experienced a power state transition (from SM to AM, or the opposite) between TS t and TS $(t-1)$, 0 otherwise. $z_s(t)$ is formally defined as:

$$z_s(t) = |O_s(t) - O_s(t-1)|, \forall s \in S, \forall t \in \mathcal{T}, \tag{10}$$

where the $|\cdot|$ denotes the absolute value operator.

We then introduce the integer variable $\rho_s(t)$ [units], which computes the total number of transitions for PS s up to TS t:

$$\rho_s(t) = \rho_s(t-1) + z_s(t), \quad \forall s \in S, \forall t \in \mathcal{T}, \tag{11}$$

where $\rho_i(t-1)$ [units] is the total number of transitions for PS s up to TS $(t-1)$.

In the following, we denote with $AF_s^{TOT}(t)$ [units] a continuous variable storing the value of AF for PS s up to TS t. The total AF is computed as in Eq. (3).

Finally, we introduce the variable $C_M^{TOT}(t)$ [$] to store the maintenance costs of the CDC at TS t. The total maintenance costs are computed as in Eq. (2).

5.2 Electricity Costs Constraints

In the following, we consider the computation of the different terms of the electricity costs. More in detail, we start by computing the CPU utilization of each PS. We denote with $u_s(t)$ [units] a continuous variable storing the CPU utilization of PS s at TS t. $u_s(t)$ [units] is expressed as the summation of the CPU consumed by the VMs running on PS s, normalized by the total CPU available on the PS. More formally, we have:

$$u_s(t) = \sum_{m \in M} x_{sm}(t) \cdot \frac{\gamma_m(t)}{\gamma_s^{MAX}}, \quad \forall s \in S, \forall t \in \mathcal{T}, \tag{12}$$

where $x_{sm}(t)$ [units] is a binary variable taking the value 1 if VM m is assigned to PS s (0 otherwise), $\gamma_m(t)$ [units] is the CPU request of VM m at TS t, and γ_s^{MAX} [units] is the maximum CPU utilization of PS s. Given the CPU utilization $u_s(t)$ [units], we then compute the total electricity costs due to CPU processing $C_E^{PROC}(t)$ [$] with Eq. (4).

In the following step, we compute the amount of data $d_{mn}^{s\sigma}(t)$ [Mb] exchanged between VM m located on PS s and VM n located on PS σ during TS t. This variable is equal to

3. The actual amount of exchanged data may be slightly higher than the size of memory, due to the retransmission of dirty memory pages. However, the typically small size of the active page set w.r.t. the global memory space of the VM allows us to neglect this effect.

the amount of data traffic $D_{mn}(t)$ [Mb] exchanged by the VMs m and n at TS t, if m and n are located on different PSs. On the other hand, if m and n are located on the same PS, $d^{s\sigma}_{mn}(t)$ is set to 0. More formally, we have:

$$d^{s\sigma}_{mn}(t) = \begin{cases} p^{s\sigma}_{mn}(t) \cdot D_{mn}(t), & \text{if} \quad s \neq \sigma \\ 0, & \text{if} \quad s = \sigma \end{cases} \quad \forall s, \sigma \in S, \forall m, n \in M, \forall t \in \mathcal{T}, \tag{13}$$

where $p^{s\sigma}_{mn}(t) = x_{sm}(t) \cdot x_{\sigma n}(t)$ is a non-linear product of decision variables. We refer the reader to Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSUSC.2018.2838338, for the detailed description of how this product is linearized. The total data transferring costs at TS t, denoted as $C^{TR}_{E}(t)$, are then defined as in Eq. (5).

In the next part, we compute the costs due to VM migrations across the PSs. Specifically, we first introduce the binary variable $y_{s\sigma m}(t)$, which takes value 1 if VM m is moved from PS s to PS σ at TS t, 0 otherwise. We set $y_{s\sigma m}(t)$ with the following constraint:

$$y_{s\sigma m}(t) = \begin{cases} q^{s\sigma}_{mn}(t), & \text{if} \quad s \neq \sigma \\ 0, & \text{if} \quad s = \sigma \end{cases} \quad \forall s, \sigma \in S, \forall m \in M, \forall t \in \mathcal{T}, \tag{14}$$

where $q^{s\sigma}_{mn}(t) = x_{sm}(t-1) \cdot x_{\sigma m}(t)$ is again a non linear product. We refer the reader to Appendix A, available in the online supplemental material, for the linearization steps.

We then store the total VM migration costs at TS t in the variable $C^{MIG}_{E}(t)$, which is defined as in Eq. (6). Finally, the total electricity costs at TS t are then computed as in Eq. (7).

5.3 Additional Constraints

We then introduce a set of additional constraints in our problem. Specifically, we first impose than each VM has to be allocated to only one PS:

$$\sum_{s \in S} x_{sm}(t) = 1, \quad \forall m \in M, \forall t \in \mathcal{T}. \tag{15}$$

Furthermore, we consider the fact that the CPU consumed by the VMs running on each PS s has to be lower than the CPU available on the PS. More formally, we have:

$$\sum_{m \in M} \gamma_m(t) \cdot x_{sm}(t) \leq \gamma^{MAX}_{s} \cdot O_s(t), \quad \forall s \in S, \forall t \in \mathcal{T}. \tag{16}$$

Similarly, we impose a limit also for the amount of memory consumed by the VMs on each PS:

$$\sum_{m \in M} \mu_m(t) \cdot x_{sm}(t) \leq \mu^{MAX}_{s} \cdot O_s(t), \quad \forall s \in S, \forall t \in \mathcal{T}. \tag{17}$$

where μ^{MAX}_{s} [Mb] is the maximum memory consumption allowed on PS s. Considering the right-hand-sides (RHSs) of the constraints (16) and (17), we remark that the presence of the products $\gamma^{MAX}_{s} \cdot O_s(t)$ and $\mu^{MAX}_{s} \cdot O_s(t)$ imposes that a VM can be assigned to a PS s in TS t, only if s is powered on in t (i.e., $O_s(t) = 1$). Indeed, if $O_s(t) = 1$, the the RHSs are equal to γ^{MAX}_{s} and μ^{MAX}_{s}, thus making the capacity available; if instead $O_s(t) = 0$, then the RHSs become 0 and no VM can be assigned to s in t.

5.4 Overall Formulation

The Optimal Maintenance and Electricity Costs (OMEC) problem is formulated as follows:

$$\min \sum_{t} C^{TOT}(t) = \sum_{t} \left[C^{TOT}_{M}(t) + C^{TOT}_{E}(t) \right]. \tag{18}$$

subject to:

$$\begin{array}{ll} \text{Maintenance Costs Computation} & (3), (2), (8) - (11), \\ \text{Electricity Costs Computation} & (4), (5), (6), (12) - (14), \\ \text{VM Allocation Constraint} & (15), \\ \text{Maximum CPU Capacity} & (16), \\ \text{Maximum Memory Capacity} & (17). \end{array} \tag{19}$$

under control variables: $x_{sm}(t) \in \{0, 1\}$, $O_s(t) \in \{0, 1\}$.

Proposition 1. *The OMEC problem is NP-Hard.*

We refer the reader to Appendix. B, available in the online supplemental material, for the proof.

6 MECDC Algorithm Description

Since the OMEC problem is very challenging to be solved even for instances of small size, we propose the Maintenance and Electricity Costs Data Center (MECDC) algorithm to practically tackle it. The main intuitions of the proposed approach are twofold: i) we do not consider all the TSs jointly together, but rather we focus on each single TS,[4] and ii) we guarantee a feasible solution which ensures the constraints (15), (16), and (17) in each TS. As a result, the MECDC algorithm is sequentially run for each TS. Specifically, for each TS τ, we use the solution computed for TS $\tau - 1$ as input for the single-period problem associated with TS τ. The solution for τ is then passed as input to the solution of the problem associated with the successive TS $\tau + 1$ and so on until we reach $\tau = |\mathcal{T}|$.

Algorithm 1 reports the MECDC pseudocode. Our solution takes inspiration from the algorithms used to solve the Bin Packing Problem [44], which are then redesigned in order to: i) take into account the different costs, and ii) considering also the impact of the solution in the long term. The algorithm requires as input the current TS index t, the CPU requirements $\gamma_m(t)$, the memory requirements $\mu_m(t)$, the amount of data transferred among the VMs $D_{mn}(t)$, as well as a matrix including the power states experienced by the PSs at previous TS. Then, MECDC produces as output the current VM to PS assignment $x_{sm}(t)$, as well as the current PSs power states. The algorithm is then divided in three main steps: i) selection of an admissible VMs to PSs allocation (lines 1-29), ii) refinement of the VMs' allocation to reduce the costs for current TS (lines 30-44), iii) adjustment of the VMs' allocation to limit the increase of the costs that will be likely experienced in the future (lines 45-51).

4. Solving the algorithm for each TS is in line with the tasks of the allocation manager detailed in Section 3.

Algorithm 1. Pseudo-Code of the MECDC Algorithm

Input: $t, \gamma_m(t), \mu_m(t), D_{mn}(t)$, prev_power_state_s
Output: $x_{sm}(t)$, curr_power_state_s
1: `Phase 1: Select an admissible VMs to PSs allocation.`
2: prev_VM_assigned=read_conf($x_{sm}(t-1)$);
3: curr_VM_assigned=prev_VM_assigned;
4: curr_power_state_s=prev_power_state_s;
5: [curr_CPU_s, curr_mem_s, flag_check]=comp_CPU_mem(curr_VM_assignment, $\gamma_m(t), \mu_m(t)$);
6: **if** flag_check==false **then**
7: s_sorted=sort(curr_CPU_s,'descend');
8: VM_sorted=sort(curr_CPU_VM,'ascend');
9: **for** vm=1:$|M|$ **do**
10: **if** check_CPU_mem(curr_VM_assigned,VM_sorted[vm], $\gamma_m(t), \mu_m(t)$)==false **then**
11: stop_condition=0;
12: **for** server=1:$|S|$ **do**
13: curr_s=s_sorted[server];
14: **if** stop_condition < 1 **then**
15: **if** curr_s != curr_VM_assigned[[VM_sorted[vm]] **then**
16: **if** check_VM_to_server(curr_VM_assigned, curr_s,VM_sorted[vm],$\gamma_m(t), \mu_m(t)$)==true **then**
17: [curr_CPU_s, curr_mem_s, curr_VM_assigned]=VM_to_server(curr_VM_assigned,curr_s,VM_sorted[vm],$\gamma_m(t)$, $\mu_m(t)$);
18: stop_condition=1;
19: **end if**
20: **end if**
21: **end if**
22: **end for**
23: **end if**
24: **end for**
25: **end if**
26: f1_VM_assigned=curr_VM_assigned;
27: [curr_total_costs, curr_power_state_s]=compute_total_costs (prev_VM_assigned, curr_VM_assigned, $t, \gamma_m(t), \mu_m(t)$, $D_{mn}(t)$, curr_power_state_s);
28: f1_total_costs=curr_total_costs;
29: f1_power_state_s=curr_power_state_s;
30: `Phase 2: Refinement of the VMs' allocation to reduce the costs for current TS.`
31: s_sorted=sort(curr_CPU_s,'ascend');
32: **for** server=1:$|S|$ **do**
33: **if** curr_power_state_s[server, t] > 0 **then**
34: [tmp_VM_assigned, flag_bin]=adaptive_bin_packing (curr_VM_assigned, s_sorted[server],$\gamma_m(t), \mu_m(t)$);
35: **if** flag_bin == 1 **then**
36: [tmp_total_costs, tmp_power_state_s]=compute_total_costs(prev_VM_assigned,tmp_VM_assignment, $t, \gamma_m(t), \mu_m(t)$, $D_{mn}(t)$, curr_power_state_s);
37: **if** tmp_total_costs $<$ curr_total_costs **then**
38: curr_VM_assigned=tmp_VM_assigned;
39: curr_total_costs=tmp_total_costs;
40: curr_power_state_s[server,t]=0;
41: **end if**
42: **end if**
43: **end if**
44: **end for**
45: `Phase 3: Adjustment of the VMs' allocation to limit the increase of the future costs.`
46: [all_on_total_costs, all_on_power_state_s, all_on_VM_assigned]=compute_total_costs_all_on (prev_VM_assigned, $t, \gamma_m(t), \mu_m(t), D_{mn}(t)$, f1_power_state_s,);
47: **if** curr_total_costs $>$ all_on_total_costs $\times\ \zeta$ **then**
48: curr_VM_assignment=all_on_VM_assignment;
49: curr_power_state_s=all_on_power_state_s;
50: **end if**
51: $x_{sm}(t)$=write_conf(curr_VM_assignment);

The first phase, that is the initial VMs' allocation (lines 1-29), is similar to the Modified Best-Fit Decreasing algorithm [10]: after the initialization of the variable for current TS (lines 2-4), the algorithm checks if the VM allocation is able to ensure the constraints (15), (16), and (17) for the current TS (line 5). If the total amount of CPU and memory requested on each PS is lower than the maximum capacity, then the algorithm passes directly to the next step, i.e., the refinement of the VMs' allocation. Otherwise, an admissible allocation needs to be selected (lines 6-25). This case may occur for example when there is the need to power on a PS that was in SM at the previous TS, in order to ensure the constraints (15), (16), and (17) for the current TS. In particular, the main intuition is to move the smallest VMs in terms of the CPU requirements to the most loaded PSs. This is done by first sorting the total CPU requested on the PSs in decreasing order (line 7), and the amount of CPU requested by each VM in increasing order (line 8). Then, in the following step (lines 10-24), the algorithm proceeds by moving the VMs from the most loaded PSs to the others, until the constraints (15), (16), and (17) are met for all the VMs and the current TS. Finally, the updated allocation of the VMs is stored (line 26), and the total costs, as well as the PS power states, are computed and saved (lines 27-29).

During the second phase, MECDC tries to find a VMs' allocation able to reduce the costs (lines 30-44), still ensuring the constraints (15), (16), and (17) for the current TS. In particular, the intuition of this part is to sort the PSs based on the amount of consumed CPU (in increasing order), and to selectively put in SM each PS, if the total costs are reduced. Initially, the PSs are sorted by increasing values of CPU (line 31). Then, for each PS in the ordered lists of PSs (line 32), if the PS is in AM (line 33), the Adaptive Bin Packing (ABP) algorithm is run (line 34), in order to migrate the VMs running on the current PS to other ones that are in AM. If the ABP algorithm succeeds (line 35), the costs of the temporary assignment are computed (line 36). If the costs are decreased compared to the current assignment (line 37), then the current assignment, the current costs, and the current power states are updated (line 38-40).

The core of phase 2 of MECDC relies on the ABP algorithm, which is detailed in Algorithm 2. This routine requires as input the current VMs to PSs allocation, the current PS from which the VMs need to be shifted, and the CPU and memory requirements. The updated VMs to PSs allocation, as well as a flag indicating the algorithm status, are produced as output. Initially (line 1), the VMs are sorted based on the amount of CPU requested. Then, the total amount of CPU and memory consumed on each PS are computed (line 2). In addition, the total number of VMs that need to be moved from the current PS, as well as the PS

power states, are stored (line 3-4). Finally, the current number of moved VMs is set to zero (line 5).

Algorithm 2. Pseudo-Code of the Adaptive Bin Packing Function

Input: curr_VM_assigned, curr_s, $\gamma_m(t), \mu_m(t)$
Output: curr_VM_assigned, flag_bin
1: [VM_sorted]=sort($\gamma_m(t)$,'ascend');
2: [curr_CPU_s, curr_mem_s, flag_check]=comp_CPU_mem (curr_VM_assignment, $\gamma_m(t), \mu_m(t)$);
3: num_VM_to_move=comp_VM_to_move(curr_VM_ assignment, curr_s);
4: power_state_s=comp_power_state_s (curr_VM_assignment);
5: VM_moved=0;
6: **for** vm=1:$|M|$ **do**
7: **if** curr_VM_assigned[VM_sorted[vm]]==curr_s **then**
8: [curr_CPU_s, curr_mem_s, flag_check]=comp_CPU_ mem(curr_VM_assignment, $\gamma_m(t), \mu_m(t)$);
9: s_sorted=sort(curr_CPU_s,'descend');
10: flag_moved_VM=0
11: **for** server=1:$|S|$ **do**
12: **if** (s_sorted[server]! =curr_s) and (flag_ moved_VM==0) and (power_state_s[s_sorted [server]] > 0) **then**
13: curr_VM_assignment[VM_sorted[vm]]=s_sorted [server];
14: [curr_CPU_s, curr_mem_s, flag_check]= comp_ CPU_mem(curr_VM_assignment, $\gamma_m(t), \mu_m(t)$);
15: **if** flag_check < 1 **then**
16: curr_VM_assignment[VM_sorted[vm]]=curr_s;
17: **else**
18: flag_moved_VM=0;
19: VM_moved++;
20: **end if**
21: **end if**
22: **end for**
23: **end if**
24: **end for**
25: **if** VM_moved==num_VM_to_move **then**
26: flag_bin=1;
27: **else**
28: flag_bin=0;
29: **end if**

In the following, the ABP iterates over the VMs (lines 6-24). If the current VM is placed on the candidate PS to be put in SM (line 7), then the algorithm tries to migrate it (lines 8-23). In particular, the total CPU requested on the PSs is computed (line 8). Then, the PSs are ordered based on the amount of requested CPU (line 9), in decreasing order. The intuition here is in fact to place the VMs on a PS which already hosts VMs, in order to limit the power state changes that may be introduced. More in depth, if the destination PS can be a candidate one (line 12), the VM is temporarily assigned to the PS (line 13). Then, the CPU and memory requirements are computed (line 14). If the constraints (15), (16), and (17) are satisfied, the current VM is allocated to the current PS, and the total number of migrated VMs is updated (line 18-19). Otherwise, the VM is kept on the original PS that was hosting it (line 16). In the last part (lines 25-29), the status flag is set. If it has been possible to move all the VMs hosted on the PS to be put in SM, then the flag is set to one (line 26). Otherwise, the flag is set to zero (line 28).

Finally, we describe the third and last phase of MECDC (lines 45-51 of Algorithm 1). The goal of this phase is to provide a mechanism to limit the potential costs growth in the future TSs. In particular, the condition of reducing the total costs at current TS (which is performed in phase 2 of the algorithm) may introduce changes in the power states of the PSs, which will have an impact on the maintenance costs paid also in the future. In order to limit this effect (without assuming the knowledge of future requirements), MECDC adopts a greedy approach, by: i) computing the total costs experienced by a solution keeping all the PSs in AM and ensuring constraints (15), (16), and (17) for the current TS (line 46), ii) checking if the total costs from the current assignment are larger than the costs of the always solution, scaled by a constant $\zeta < 1$, iii) setting the current allocation to the always on allocation in case the condition at ii) occurs. As for i) we compute the VMs to PSs allocation with the Next Fit Decreasing (NFD) algorithm reported in Appendix D, available in the online supplemental material, which tends to keep all the PSs always powered on, in order to balance the CPU load across the PSs.

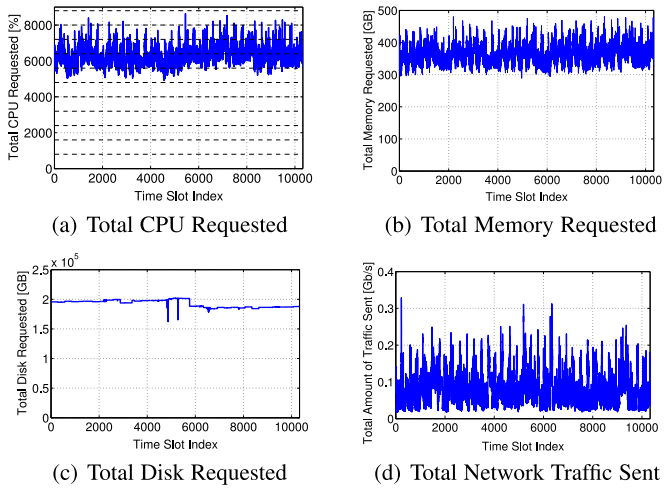
6.1 Complexity Analysis

We analyze the time complexity of MECDC. Focusing on the first phase (lines 2-25), the computation of CPU and memory requirements on each PS is done in $\mathcal{O}(|M| \cdot |S|)$ iterations (line 5). Similarly, checking if a given VM can be migrated to a given PS (lines 16), as well as the VM migration (line 17), can be done in $\mathcal{O}(|M| \cdot |S|)$ iterations. The procedure is then repeated for each VM (line 9) and each PS (line 12) in the worst case. As a result, the overall complexity of phase 1 is $\mathcal{O}(|M|^2 \cdot |S|^2)$.

Focusing then on the second phase of MECDC (lines 30-44), the ABP algorithm is run on each PS in the worst case. Therefore, it is necessary to estimate the complexity of the ABP routine. In particular, the preliminary steps of ABP (lines 1-5 of Algorithm 2) require $\mathcal{O}(|M|(\log|M| + |S|))$ iterations. In addition, the computation of CPU and memory requirements (lines 8,14 of Algorithm 2) requires $\mathcal{O}(|M| \cdot |S|)$ iterations. This computation is potentially repeated for each PS (line 11 of Algorithm 2) and each VM (line 6 of Algorithm 2). As a result, the overall complexity of ABP is in the order of $\mathcal{O}(|M|^2 \cdot |S|^2)$. Going back to the second phase of MECDC, the ABP algorithm is potentially repeated for each PS (line 34). In addition, the computation of the total costs requires $\mathcal{O}(|M|^2 \cdot |S|^2)$ iterations. Overall, the complexity of the second phase is in the order of $\mathcal{O}(|M|^2 \cdot |S|^3)$.

Focusing on the third phase of MECDC, this steps requires: i) the computation of an always on allocation, ii) the computation of the costs for this allocation. Focusing on i), we adopt the NFD algorithm, whose complexity (reported in Appendix 4, available in the online supplemental material) is in the order of $\mathcal{O}(|M|^2 \cdot |S|)$. Focusing on ii), this part can be performed in $\mathcal{O}(|M|^2 \cdot |S|^2)$ iterations. As a result, the overall complexity of MECDC, i.e., from the start to the end, is in the order of $\mathcal{O}(|M|^2 \cdot |S|^3)$. Even though this complexity may appear relatively high at a first glance, it is pretty limited in realistic scenarios (which are going to be described in Section 7), due to the fact that it is necessary to keep the power states of the PSs unchanged in most of TSs, in order to satisfy the CPU and memory requirements, as well as limiting the impact of maintenance costs.

Finally, we analyze the space complexity of MECDC. Overall, this solution requires temporary arrays of size $|M|$



(a) Total CPU Requested (b) Total Memory Requested

(c) Total Disk Requested (d) Total Network Traffic Sent

Fig. 2. Evolution of the CPU, memory, disk, and network traffic versus TS index for the considered DC trace.

and $|S|$. The same applies also to the ABP routine. In addition, the power states experienced by the PSs during the past TSs are required, resulting in a matrix of size $|S| \cdot |T|$. Finally, the algorithm requires a matrix of size $|S| \cdot |M|$ to store the VM to PS assignment, as well as a matrix of updated power states, whose size is $|S| \cdot |T|$. As a result, the overall space complexity is in the order of $\mathcal{O}(|S| \cdot (|M| + |T|))$.

7 Scenarios and Input Parameters

When evaluating an algorithm, selecting a meaningful and realistic set of input parameters is of crucial importance. To pursue this goal, we have considered a set of realistic traces to provide the VMs-related input parameters. In addition, we have taken from previous works, as well as from the analysis of the realistic traces, the input parameters for the PS set. The following sections detail the pursued methodology.

7.1 Virtual Machines Parameters

The considered parameters for each VM m and for each TS t include: the requested CPU $\gamma_m(t)$, ii) the requested memory $\mu_m(t)$, iii) the amount of data $D_{mn}(t)$ exchanged by the VM m to each other VM $n \in M$. In order to retrieve such parameters, we have considered the trace Materna-3, which reports real measurements of a CDC collected from TU Delft [45], [46], [47]. The trace includes the log files of 547 VMs, which are used to deploy a CDC devoted to business intensive applications. Each VM log reports a set of information collected for each TS, including: i) the CPU requirements (both in terms of CPU percentage and in terms of CPU cores), ii) the memory requirement, iii) the amount of disk provisioned to the VMs, iv) the total amount of traffic sent out from the VMs. The time granularity of the collected log entries is $\delta(t) = 5$ [minutes],[5] for a period of around 5 weeks in total, measured during the year 2016. To give more insight, Fig. 2 reports the evolution over time of the following consolidated metrics: i) total amount of CPU requested, ii) total amount of memory requested, iii) total amount of disk provisioned, iv) total amount of network traffic sent. Interestingly, both the total CPU (Fig. 2a) and the total memory requirements (Fig. 2b) tend to notably vary over time, with peaks that suggests a daily and weekday periodicity. On the other hand, the total amount of disk provisioned (Fig. 2c) is pretty constant. Finally, the total amount of traffic sent (Fig. 2d) is also experiencing a notable variability.

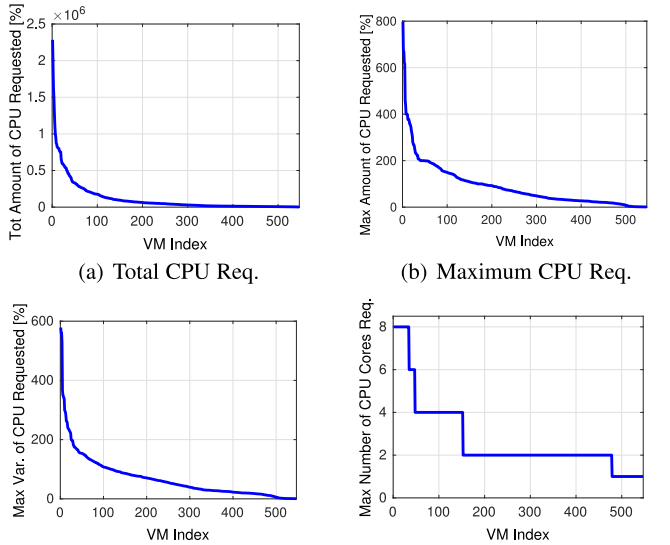
Given the available trace information, and the fact that there is a remarkable variation of CPU and memory over time, a natural question is then: is it possible to extract meaningful set of VMs with common features, in order to test the proposed algorithm? Indeed, our goal is not only to evaluate the impact of the proposed solution on the whole trace available, but also to generalize our findings to typical cases, that can be representative of different classes of VMs. In order to tackle this issue, we have focused on the amount of CPU requested by each VM, which can be one of the typical feature to classify the VMs. In particular, we have computed for each VM the following metrics over the whole trace: i) total amount of requested CPU, ii) maximum amount of requested CPU, iii) maximum variation of CPU, which is expressed as $\max_t |\gamma_m(t) - \gamma_m(t-1)|$ for each VM, iv) maximum number of requested CPU cores. For each metric, we have then sorted the VMs in decreasing order. Fig. 3 reports the obtained results. Interestingly, the metrics reveal a strong heterogeneity among the VMs, with trends similar to power-laws, especially in Figs. 3a, 3b, and 3c. Given these trends, we have therefore selected four representative subsets of $|\mathcal{M}| = 15$ VMs for each metric, namely: Tot-CPU, Max-CPU, MaxVar-CPU, MaxCores-CPU. In particular, we have selected the most demanding VMs for each considered metric, in order to test our algorithm under different conditions.

Fig. 4 reports the total CPU variation for each VM subset over time. Interestingly, we can note that there are four distinct patterns emerging from the subset. In particular, the total CPU is maximized by the Tot-CPU pattern (as expected). On the other hand, both the Max-CPU and MaxVar-CPU subset require less CPU, but are more subject to strong CPU oscillations. Finally, the MaxCores-CPU subset is the least demanding in terms of total CPU. This is due to the fact that a VM provisioned with a large number of cores does not necessarily use all the available CPU resources.

To give more insight, we have analyzed if the same VMs are included in the different subsets. To this aim, Table 1 reports the obtained confusion matrix. As expected, the majority of VMs in the Tot-CPU subset does not appear in the other ones. The same applies also to the MaxCores-CPU subset. Finally, different VMs are shared between the Max-CPU and the MaxVar-CPU subsets. Thus, we can conclude that the selected subsets: i) include different VMs, ii) are representative of different trends.

Up to now, we are able to set the requested CPU $\gamma_m(t)$ and the requested memory $\mu_m(t)$ directly from the trace data. Focusing then on the amount of traffic exchanged by the VMs, the available trace only includes information about the total traffic sent by each VM, namely, $\sum_n D_{mn}(t)$. Therefore, the single values of $D_{mn}(t)$ needs to be retrieved in some manner. To do that, we proceed as follows: i) when we consider the whole CDC, we assume that 80 percent of traffic is sent to the 20 percent of VMs that are actually sending the largest amount of data; the remaining 20 percent of traffic is uniformly distributed among the remaining VMs; ii) when we consider the subsets of $|M| = 15$ VMs, we assume that the $\sum_n D_{mn}(t)$ traffic of each VM is uniformly distributed across the remaining $|M| - 1$ VMs.

5. The TS duration is an input parameter of MECDC. This parameter does not impact the time complexity of our algorithm. Lower durations should be set in accordance to the amount of time required to change the PS power state. Higher durations would make MECDC less reactive in terms of both migrations and PS power state changes.



(a) Total CPU Req. (b) Maximum CPU Req.

(c) Maximum Var. of CPU Req. (d) Max. Number of Cores Req.

Fig. 3. VMs ordered according to different rules (Note: the VM indexes change between one ordering rule and another one).

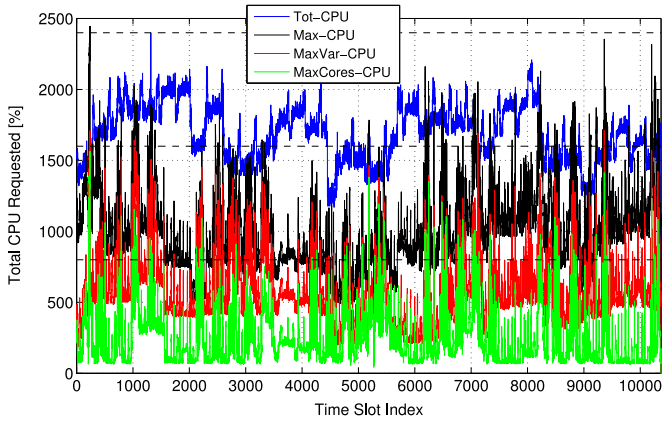


Fig. 4. Total Variation of CPU versus the TS index for the different VM subsets.

Finally, we repeat the measured trace over a total period of time $\mathcal{T} = 5$ [years]. In this way, we consider an amount of time sufficiently long to evaluate the impact of the maintenance costs, which tend to be increased in the long term.

7.2 Physical Servers Parameters

We then consider the parameters related to PSs. In particular, by observing the maximum number of CPU cores requested by each VM (see Fig. 3d), we assume that each PS has 8 cores, each of them able to be used up to 100 percent. As a result, $\gamma_s^{MAX} = 800$ [unit] for each PS s. Moreover, each PS is equipped with a large amount of memory, i.e., $\mu_s^{MAX} = 128$ [GB]. Clearly, a question now arises: how many PSs should be deployed in the scenario? To answer this question, the dashed lines of Fig. 2a are drawn every $\gamma_s^{MAX} = 800$ [units] of CPU. In order to satisfy the maximum CPU requirements, we can easily see that no less that 11 PSs needs to be deployed. However, due to the fact that each VM request cannot be split across multiple PS, it is necessary to add an amount of spare capacity to practically fulfil the CPU requests. In our case, we have found that by setting the number of PSs $|S|$ equal to 14 it is possible to always ensure both the CPU and memory requirements. In addition, we point out

TABLE 1
Confusion Matrix Reporting the Number of Same VMs across the Different Subsets

VM Subset	Tot-CPU	Max-CPU	MaxVar-CPU	MaxCores-CPU
Tot-CPU	15	4	3	0
Max-CPU	4	15	11	3
MaxVar-CPU	3	11	15	3
MaxCores-CPU	0	3	3	15

that disk requirements are less stringent, due to the fact that: i) the disk requirements do not strongly vary over time, ii) it is feasible and not cost expensive to over-provision the PSs with large disks, and iii) it is a common practice to store on the physical PS disk just the operating system for the hypervisor, while the VMs images are stored in a separate Network Attached Storage. Eventually, also the amount of data sent from the VMs is globally lower than the capacity of available network connections, which is currently in the order of [Gbps]. Finally, a similar procedure is repeated also for the different subsets of VMs, ending that the setting $|S| = 4$ is able to always fulfil all the requirements from the VMs.

In the following, we focus on the power and energy cost parameters. We set the maximum and minimum PS power equal to $P_s^{MAX} = 328.2$ [W] and $P_s^{IDLE} = 197.6$ [W], respectively, in accordance to measurement provided by [48]. The interface power P_s^{TR-IF} is set equal to 42.7 [W] [49], [50]. The power due to data transferring $P_{s\sigma}^{TR-NET}$ is set equal to 0.003 [W/b] if $s \neq \sigma$, 0 otherwise, in accordance to [51]. The power due to overhead P_s^{OH} is set equal to 1 percent of P_s^{MAX} [51]. Focusing on the cost parameters, we set the electricity costs K_E equal to 0.00016 [$/Wh] [13].

In the next part, we focus on the parameters related to maintenance operations, namely: i) the AF in SM AF_s^{SM}, ii) the weight for power state transitions Ψ_s, iii) the FR of the PS always in AM ϕ_s^{AM}, and iv) the cost for a single reparation K_R. We then detail in the following the setting of each parameter. In order to set the AF in SM AF_s^{SM}, we recall that this term is equal to ϕ_s^{SM}/ϕ_s^{AM}, where ϕ_s^{SM} [1/h] is the FR in SM, which is expressed by the Arrhenius law [52]:

$$\phi_s^{SM} = e^{\frac{-E_a}{(\mathcal{K} \cdot T_{SM})}}, \tag{20}$$

where E_a [joule/mol] is the activation energy, $\mathcal{K} = 8.314472$ [joule / (mol kelvin)] is the Boltzmann constant, and T_{SM} [kelvin] is the temperature in SM. In our case, we have set $E_a = 30500$ [joule/mol] in accordance to the values measured for chip components in [53], $T_{SM} = 303.15$ [kelvin], corresponding to 30 [Celsius], in accordance to the real measurement performed on a PS in [14]. As a result, we get $AF_s^{SM} \approx 0.5$, which is used for each PS $s \in S$. In the following, we focus on the FR of the PS ϕ_s^{AM} always in AM. In particular, we set the FR of the PS $\phi_s^{AM} = 1.14 \times 10^{-5}$ [1/h] $\forall s \in S$ [13]. In the following, we focus on the setting of the Ψ_s parameter. More in depth, Ψ_s is defined as $\Psi_s = 1/(\phi_s^{AM} N_s^F)$ where N_s^F is the number of cycles to failures. In our case, we consider the interval $N_s^F = [8.77 \cdot 10^5\text{-}8.77 \cdot 10^6]$. In particular, we set N_s^F to values higher than the ones measured under stressful conditions, i.e., between a maximum and a minimum temperature (such as the testing methodology of [54]), due to the fact that we are only applying a SM procedure, which is supposed to be less aggressive for the lifetime of the components than the test in [54]. As a result, we consider a range of Ψ_s values in

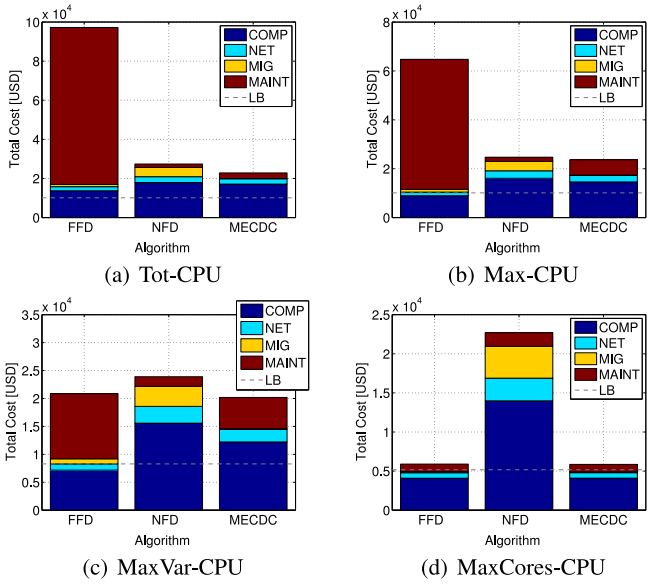


Fig. 5. Costs breakdown at the end of the considered 5-years periods across the different VMs subsets considering MECDC, FFD, NFD, and the Lower Bound (LB).

the interval $[0.01\text{-}0.1]$. Finally, the reparation cost for one PS K_R is set equal to 380 [$] [13].

8 Performance Evaluation

We evaluate the performance of MECDC against two reference algorithms, namely First Fit Decreasing (FFD) and a modified version of the Next Fit Decreasing (NFD). We refer the reader to Appendix C, available in the online supplemental material, and Appendix D, available in the online supplemental material, for a detailed description of FFD and NFD, respectively. In brief, the main goal of FFD is to approximate the Bin Packing Problem, in order to limit the number of used PSs, and therefore the associated processing costs. On the other hand, the NFD algorithm aims to keep all the PSs always powered on, and to reduce the load on each PS by distributing the VMs across the set of PSs. Similarly to MECDC, both FFD and NFD compute the set of PSs powered on and the VM to PS assignment in each TS.

Apart from the reference solutions, we consider also a Lower Bound (LB) to assess better the positioning of our approach. We refer the reader to Appendix E, available in the online supplemental material, for the detailed steps of the LB computation. In brief, the LB is able to assess the minimum processing and maintenance costs that need to be paid in any case, in order to satisfy the VMs requirements in terms of CPU and memory.

We code MECDC, FFD, NFD and LB in Matlab v. 2012, and we run them on a Linux Desktop PC, equipped with an Intel Core I5 processor and 8 [GB] of RAM.

8.1 Impact of the VM Subsets

We first run the strategies over the different subsets of VMs, by considering the set of the parameters reported in the previous section, and a value of $\Psi_s = 0.06$ for all the scenarios. Moreover, we set the ζ parameter of MECDC to 0.5.[6] Fig. 5 reports the total costs incurred by summing the costs from all the TSs. Moreover, each subfigure details for each algorithm the cost components, namely processing (COMP), data transferring across the DC network (NET), migrations (MIG) and maintenance (MAINT). As expected, the FFD algorithm tends to achieve the lowest processing costs across all scenarios. However, reducing the processing costs is not always beneficial for the maintenance costs, as shown e.g., in Figs. 5a, 5b, and 5c. In particular, due to the fluctuations of the CPU requests, there are cases in which FFD introduces a lot of transitions in the power states of the PSs, resulting in a large increase of the maintenance costs. On the other hand, the maintenance costs are reduced by the NFD solution, which tends to keep all the PSs always powered on. However, keeping the PSs always powered on generally results in large inefficiencies, as shown, e.g, in Fig. 5d. Finally, we can note that the proposed MECDC is able to leverage the tradeoff between all the costs, and to achieve the best solutions compared to NFD and FFD. Moreover, MECDC is always pretty close to the LB in all the considered scenarios. In particular, the good performance of MECDC is realized by means of: i) the analysis of all the involved costs before taking a decision involving the PS power states and the VM migrations, ii) the introduction of a safety mechanism to limit the increase of the maintenance costs in the long term.

In the following, we analyze the transient behavior of the considered algorithms, by computing the average cost per TS. In particular, the average cost for each TS t is computed by averaging the total cost over the TS 1 and t. Fig. 6 reports the costs for the different algorithms across the different subset of VMs. By observing the trends reported in the subfigures, we can note that the average cost of NFD tends to be always constant. This is an expected result, since this solution tends to keep always the PS powered on, and therefore to not vary consistently both the processing costs and the maintenance ones. On the other hand, the average costs of MECDC and NFD tends to vary with time. More in detail, during the initial months, the costs per TS is generally lower for both FFD and MECDC compared to NFD. This is due to the fact both these solutions are able to vary the power states of the PS, and consequently to decrease the processing costs. However, the costs of the FFD solution are generally increasing with time, and even surpass the costs of MECDC in the Tot-CPU (Fig. 6a) and Max-CPU subsets (Fig 6b). Actually, FFD is completely agnostic of the impact of PS transitions, which not only primarly affect the maintenance costs, but have also an impact on migrations and data transferring costs. The only case in which the trend of FFD is pretty constant is the MaxCores-CPU (Fig. 6d). By further investigating this fact, we have found that for this subset the number of PSs is over-dimensioned. As a result, it is always possible to keep powered off different PSs across the TSs, and to limit the number of PS transitions. Finally, we can note that in this case MECDC is pretty close to FFD. Summarizing, MECDC is able to keep a balanced solution, and the achieve the lowest average costs at the end of the considered time period.

8.2 Sensitivity Analysis

In the following, we focus on the Tot-CPU subset and we consider the variation of the main input parameters. We start with the variation of the number of PSs, as reported Table 2. In particular, we can note that the increase of the number of

6. We have performed a sensitivity analysis (not reported here due to the lack of space), finding that the setting $\zeta = 0.5$ provides good performance in all the scenarios.

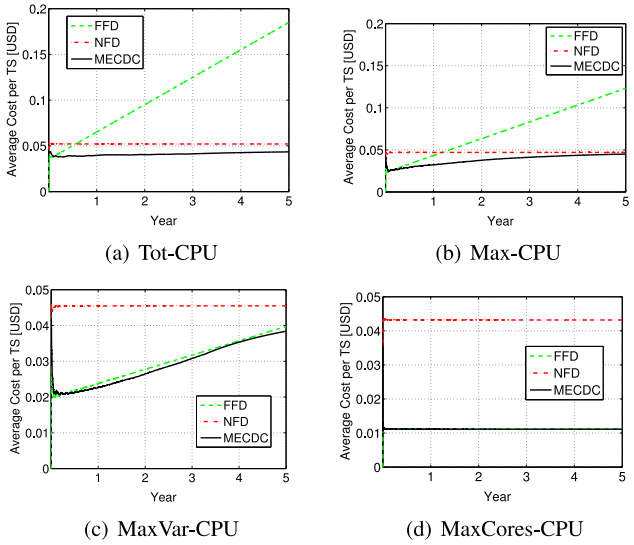


Fig. 6. Average cost per TS versus time across the different VMs subsets considering MECDC, FFD, and NFD.

PSs tends to increase the total costs of NFD, due to the increase in the number of PSs powered on. Clearly, introducing more PSs tends also to increase the costs obtained by the LB. Focusing then on FFD, the passage from $|S| = 4$ to $|S| = 5$ is able to notably reduce the number of PS transitions, and consequently to decrease the total costs. However, we can note that the costs tends to slightly increase for higher values of $|S|$. Finally, the MECDC solution always achieve the best solution after the LB. In particular, MECDC is able to save between [782-74397] [$] compared to FFD, and between [4546-23520] [$] compared to NFD.

We then continue our analysis by varying the Ψ_s parameter, which governs the impact of power state transitions on the costs. From the observations reported in Section 7.2, a reasonable range for this parameter is between 0.01 and 0.1. We therefore rerun the algorithms and the LB by considering a variation of the Ψ_s parameter in this range. Table 3 reports the results obtained over the Tot-CPU subset. More in detail, the NFD algorithm is not affected by Ψ_s, since the PS power state is kept unchanged by this solution. Similarly, also the LB does not vary, since the impact of PS transitions is not taken into account. On the other hand, the increase of Ψ_s has a great impact on FFD, whose costs tends to largely increase and largely surpass the ones of NFD. Interestingly, MECDC experiences a modest increase in the total costs, and it is always the best solution compared to NFD and FFD. As a result, MECDC is robust against any variation of Ψ_s in the considered range.

In the following part, we analyze the impact of the ζ parameter on the performance of MECDC. We recall that this parameter is used to adopt an always on solution when the current costs are higher that the costs of the always on solution, scaled by the ζ parameter. Table 4 reports the different components of the costs versus the variation of ζ. By observing the trend of the different components when ζ is increased, we can note that: i) the processing costs $\sum_t C_E^{PROC}$ tend to decrease, ii) the data transferring costs $\sum_t C_E^{TR}$ are decreased, iii) both migrations $\sum_t C_E^{MIG}$ and maintenance $\sum_t C_M^{TOT}$ are increased. These trends are due to the fact that, when ζ is close to 0, the algorithm tends to frequently apply the always on solution, which results in an increase of the data processing and transferring costs, while

TABLE 2
Total Costs versus the Number of PSs $|S|$ for the Different Strategies (Tot-CPU Subset)

	FFD	NFD	MECDC	LB
$\|S\| = 4$	97230 [$]	27379 [$]	22833 [$]	14800 [$]
$\|S\| = 5$	18165 [$]	31912 [$]	17383 [$]	15016 [$]
$\|S\| = 6$	18407 [$]	36383 [$]	17092 [$]	15231 [$]
$\|S\| = 7$	18649 [$]	40836 [$]	17316 [$]	15447 [$]

TABLE 3
Total Costs versus the Variation of the Ψ_s Parameter for the Different Strategies (Tot-CPU Subset)

	FFD	NFD	MECDC	LB
$\Psi_s = 0.01$	31057 [$]	27379 [$]	20489 [$]	14800 [$]
$\Psi_s = 0.03$	57526 [$]	27379 [$]	21994 [$]	14800 [$]
$\Psi_s = 0.06$	97230 [$]	27379 [$]	22833 [$]	14800 [$]
$\Psi_s = 0.08$	123700 [$]	27379 [$]	23265 [$]	14800 [$]
$\Psi_s = 0.1$	150170 [$]	27379 [$]	23862 [$]	14800 [$]

TABLE 4
Cost Breakdown versus the Variation of the ζ Parameter for the MECDC Strategy (Tot-CPU Subset)

	$\sum_t C_E^{PROC}$	$\sum_t C_E^{TR}$	$\sum_t C_E^{MIG}$	$\sum_t C_M^{TOT}$	$\sum_t C^{TOT}$
$\zeta = 0.1$	17912 [$]	2965 [$]	1.78 [$]	1725 [$]	22605 [$]
$\zeta = 0.5$	17096 [$]	2798 [$]	3 [$]	2935 [$]	22833 [$]
$\zeta = 0.7$	14855 [$]	2333 [$]	46 [$]	5819 [$]	23054 [$]
$\zeta = 1.0$	13504 [$]	2045 [$]	109 [$]	6301 [$]	21960 [$]
$\zeta = 1.5$	13505 [$]	2045 [$]	112 [$]	6305 [$]	21965 [$]

reducing both migrations and maintenance ones. On the other hand, when ζ is increased, the algorithm is more prone to PS transitions, resulting in the opposite effects on the costs. Although the total costs are less impacted by the ζ variation in this case, we believe that this parameter can be useful for the content provider in order to tune the algorithm to the specific scenario considered. For example, in cases where it is important to reduce the amount of migrations (e.g., to reduce the associated delay), the ζ parameter should be set to low values (i.e., $\leq$0.1).

Finally, we have considered the impact of varying the TS duration $\delta(t)$. Results, reported in Appendix G, available in the online supplemental material, confirm that MECDC is always able to guarantee the lowest costs compared to NFD and FFD.

8.3 Impact of the Heterogeneity of Physical Servers

Up to this point, a natural question is then: what is the impact when different classes of PSs are taken into account? To investigate this issue, we have considered a set of heterogeneous PSs, as reported in Table 5. In particular, we have considered two categories of PSs, having different power requirements, and different CPU capacities. The remaining PS parameters are the same as in the previous experiments.

We have then run the algorithms over the heterogeneous PSs scenario and the Tot-CPU subset of VMs. Fig. 7 reports the breakdown of the total costs at the end of the 5-years period, while Table 6 reports the costs in terms of values. Although FFD is able to reduce the total costs compared to

TABLE 5
Server Features for the Heterogeneous Scenario [48]

Server Index	P_s^{MAX}	P_s^{IDLE}	γ_s^{MAX}
1-2	270.8 [W]	138.9 [W]	400 [units]
3-6	328.2 [W]	197.6 [W]	800 [units]

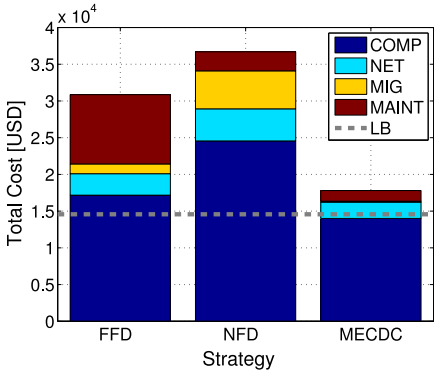


Fig. 7. Costs breakdown at the end of the considered 5-years period for the Tot-CPU subset and the heterogeneous PSs set considering MECDC, FFD, NFD, and the Lower Bound (LB).

TABLE 6
Total Costs for the Different Strategies
(Tot-CPU Subset and Heterogeneous PSs Set)

FFD	NFD	MECDC	LB
30860 [$]	36700 [$]	17804 [$]	14576 [$]

NFD, the best algorithm is MECDC, which is able to notable reduce the costs, being also close to the lower bound. The good performance of MECDC is due to the fact this solution explicitly takes into account all the costs, including the ones arising from the different values of PS power consumption.

8.4 Analysis on the Entire DC

In the last part of our work, we have run the MECDC, NFD and FFD algorithms over the All-DC set, which we recall is composed of 547 VMs and 14 PSs. Table 7 reports the details for each cost component across the different strategies. Interestingly, MECDC is able to achieve the lowest cost in each component, or being very close to the lowest values. Compared to the previous scenarios, in which we considered 15 VMs and 4 PSs, in the All-DC set the number of VMs is increased by 38 times, while the number of PSs by a factor of 3.5. As a result, the impact of migrations is much larger, since a larger number of VMs is hosted in each PS. Even in this scenario, MECDC guarantees the best performance. In particular, MECDC is able to reduce the migrations costs by a factor between 64 and 82 percent compared to the other solutions. Considering then the total costs, MECDC is able to save 59930-252390 [$] compared to NFD/FFD.

Finally, we have analyzed the average computation time per TS for the different algorithms over the All-DC set, as reported in Table 8. As expected, MECDC requires more time to retrieve a solution compared to NFD and FFD. However, the average computation time is lower than 1.3 [s], a number that is much lower compared to the TS duration, which is in the order of minutes in our considered scenarios. As a result, we can conclude that MECDC is also very effective in limiting the required computation time.

TABLE 7
Cost Breakdown for the Different Strategies (All-DC Set)

	$\sum_t C_E^{PROC}$	$\sum_t C_E^{TR}$	$\sum_t C_E^{MIG}$	$\sum_t C_M^{TOT}$	$\sum_t C^{TOT}$
FFD	45599 [$]	8854 [$]	107430 [$]	3140 [$]	165020 [$]
NFD	60296 [$]	11975 [$]	279600 [$]	5608 [$]	357480 [$]
MECDC	44895 [$]	8724 [$]	48330 [$]	3145 [$]	105090 [$]

TABLE 8
Average Computation Time Per TS
for the Different Strategies (All-DC Set)

FFD	NFD	MECDC
0.32 [s]	0.33 [s]	1.28 [s]

8.5 Discussion

Our work points out the necessity of a joint approach for balancing the electricity consumption and the maintenance costs in a CDC. This becomes evident when the amount of time under consideration is in the order of years, as the fatigue effects are experienced only when a PS repeatedly changes its power states. Why has such effect not been considered in the literature so far? The answer is that actually the energy consolidation algorithms, and in general the solutions targeting the reduction of energy consumption, are more concentrated on the most evident (and prompt) effect, which is the power variation over time. Therefore, in order to reduce the power consumption, it makes sense to optimize the PS power states even with policies that take power state decisions at each TS. As we have shown in this work, reducing solely the electricity costs is not wise in the long period (see e.g., the FFD strategy results reported in Figs. 5a, 5b, and 5c), since the maintenance costs are increased to a large extent. On the other hand, we point out that the proposed MECDC solution is always able to wisely balance between the electricity consumption and the maintenance costs. Clearly, MECDC has a higher computation complexity compared to the algorithms focused solely on electricity consumption. However, our results show that MECDC allows to retrieve a solution in a reasonable amount of time (less than 2 [s] for each TS), even for large DCs composed of hundreds of VMs.

Another aspect potentially affecting the results is the delay introduced by the migration of VMs across PSs. This aspect is not explicitly addressed in this work, since we assume that live migrations can be performed without impacting the VM delay requirements. However, in Appendix F, available in the online supplemental material, we provide a first evaluation of the impact from considering the VM delay constraints, and how the MECDC algorithm is modified to integrate them.

9 Conclusions and Future Directions

We have targeted the problem of jointly managing the maintenance costs and the electricity consumption in a CDC. After showing that changing the power states of PSs has an impact on both the failure management costs, as well as the energy consumption, we have formulated the OMEC problem, with the goal of jointly managing the aforementioned costs. Since the OMEC problem is NP-Hard, we have described the MECDC algorithm, which has been designed to wisely leverage the tradeoff between different costs, as well as taking into account their long term impact over time. Results, obtained over a set of realistic scenarios, clearly show that MECDC

always requires consistently lower costs compared to the FFD and NFD reference algorithms. Moreover, we have also shown that the total costs obtained by MECDC are also close to a lower bound. In addition, the computation time, obtained from a scenario in which there are hundreds of VMs and by running the algorithm on a Desktop PC, is very low, i.e., less than 2 [s] on average.

As next steps, we plan to face different issues, including: i) the definition and evaluation of more complex failure models to take into account the impact on different components, as well as different temperatures of CPU cores, ii) the introduction of delay costs for migrating VMs across PSs, iii) the application of our approach to a set of CDCs, each of them subject to different electricity prices (e.g., due to different CDC locations).

Acknowledgments

The authors would like to thank the reviewers for their valuable feedback.

References

[1] H. Wells, *World Brain*. London, United Kingdom: Methuen & Co., 1938.
[2] V. Bush, et al., "As we may think," *The Atlantic Monthly*, vol. 176, no. 1, pp. 101–108, 1945.
[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.
[4] P. Mell, T. Grance, et al., "The NIST definition of cloud computing," [Online]. Available: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf, Last Accessed on: June 5, 2018.
[5] M. Hilbert and P. López, "The worlds technological capacity to store, communicate, and compute information," *Sci.*, vol. 332, no. 6025, pp. 60–65, 2011.
[6] C. D. Patel and A. J. Shah, "Cost model for planning, development and operation of a data center," HP Laboratories, Palo Alto, CA, Tech. Rep. HPL-2005-107(R.1), 2005.
[7] J. Koomey, "Growth in data center electricity use 2005 to 2010," [Online]. Available: http://www.twosidesna.org/download/Koomey_Johnathon_G-_Growth_In_Data_Center_Electricity_Use_2005_to_2010_2011.pdf, Last Accessed on: June 5, 2018.
[8] K. Bilal, S. U. R. Malik, O. Khalid, A. Hameed, E. Alvarez, V. Wijaysekara, R. Irfan, S. Shrestha, D. Dwivedy, M. Ali, et al., "A taxonomy and survey on green data center networks," *Future Generation Comput. Syst.*, vol. 36, pp. 189–208, 2014.
[9] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proc. Conf. Power Aware Comput. Syst.*, pp. 1–5, vol. 10, 2008.
[10] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.
[11] C. Mastroianni, M. Meo, and G. Papuzzo, "Probabilistic consolidation of virtual machines in self-organizing cloud data centers," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 215–228, Jul.-Dec. 2013.
[12] W. Fang, X. Liang, S. Li, L. Chiaraviglio, and N. Xiong, "VMPlanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers," *Comput. Netw.*, vol. 57, no. 1, pp. 179–196, 2013.
[13] L. Chiaraviglio, P. Wiatr, P. Monti, J. Chen, J. Lorincz, F. Idzikowski, M. Listanti, and L. Wosinska, "Is green networking beneficial in terms of device lifetime?," *IEEE Commun. Mag.*, vol. 53, no. 5, pp. 232–240, May 2015.
[14] L. Chiaraviglio, N. Blefari-Melazzi, C. Canali, F. Cuomo, R. Lancellotti, and M. Shojafar, "A measurement-based analysis of temperature variations introduced by power management on commodity hardware," in *Proc. Int. Conf. Transparent Opt. Netw.*, 2017, pp. 1–4.
[15] D. Frear, D. Grivas, and J. Morris, "Thermal fatigue in solder joints," *JOM*, vol. 40, no. 6, pp. 18–22, 1988.
[16] W. Lee, L. Nguyen, and G. S. Selvaduray, "Solder joint fatigue models: review and applicability to chip scale packages," *Microelectron. Rel.*, vol. 40, no. 2, pp. 231–244, 2000.
[17] R. P. G. Muller, *An Experimental and Analytical Investigation on the Fatigue Behaviour of Fuselage Riveted Lap Joints. The Significance of the Rivet Squeeze Force, and a Comparison of 2024-T3 and Glare 3*, The Netherlands: Delft Univ. Technology, 1995.
[18] J. Mi, Y.-F. Li, Y.-J. Yang, W. Peng, and H.-Z. Huang, "Thermal cycling life prediction of Sn-3.0 Ag-0.5 Cu solder joint using type-I censored data," *Sci. World J.*, vol. 2014, 2014, Art. no. 807693.
[19] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Commun. Surveys Tutorials*, vol. 18, no. 1, pp. 732–794, Jan.-Mar. 2016.
[20] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2009, pp. 254–265.
[21] J. Stoess, C. Lang, and F. Bellosa, "Energy management for hypervisor-based virtual machines," in *Proc. USENIX Annu. Tech. Conf.*, 2007, pp. 1–14.
[22] H. Liu, H. Jin, C.-Z. Xu, and X. Liao, "Performance and energy modeling for live migration of virtual machines," *Cluster Comput.*, vol. 16, no. 2, pp. 249–264, 2013.
[23] G. Soni and M. Kalra, "A novel approach for load balancing in cloud data center," in *Proc. IEEE Int. Advance Comput. Conf.*, 2014, pp. 807–812.
[24] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proc. 10th IEEE/ACM Int. Conf. Cluster Cloud Grid Comput.*, 2010, pp. 826–831.
[25] J. Bi, Z. Zhu, R. Tian, and Q. Wang, "Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center," in *Proc. IEEE 3rd Int. Conf. Cloud Comput.*, 2010, pp. 370–377.
[26] R. Han, M. M. Ghanem, L. Guo, Y. Guo, and M. Osmond, "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications," *Future Generation Comput. Syst.*, vol. 32, pp. 82–98, 2014.
[27] J. L. Berral, R. Gavalda, and J. Torres, "Power-aware multi-data center management using machine learning," in *Proc. 42nd Int. Conf. 42nd Int. Conf. Parallel Process.*, 2013, pp. 858–867.
[28] R. Ge, X. Feng, and K. W. Cameron, "Modeling and evaluating energy-performance efficiency of parallel processing on multicore based power aware systems," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2009, pp. 1–8.
[29] S. L. Song, K. Barker, and D. Kerbyson, "Unified performance and power modeling of scientific workloads," in *Proc. 1st Int. Workshop Energy Efficient Supercomputing*, 2013, Art. no. 4.
[30] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, 2008.
[31] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 75–86, 2008.
[32] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 51–62, 2009.
[33] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 63–74, 2009.
[34] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. U. Khan, "GreenCloud: A packet-level simulator of energy-aware cloud computing data centers," in *Proc. Global Telecommun. Conf.*, 2010, pp. 1–5.
[35] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, pp. 350–361, 2011.
[36] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, pp. 193–204.
[37] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable virtual data center embedding in clouds," in *Proc. IEEE INFOCOM*, 2014, pp. 289–297.
[38] J. Fan, C. Guan, Y. Zhao, and C. Qiao, "Availability-aware mapping of service function chains," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[39] R. Jhawar and V. Piuri, "Fault tolerance management in IaaS clouds," in *Proc. IEEE 1st AESS Eur. Conf. Satellite Telecommun.*, 2012, pp. 1–6.
[40] J. Lau, W. Danksher, and P. Vianco, "Acceleration models, constitutive equations, and reliability of lead-free solders and joints," in *Proc. 53rd Electron. Compon. Technol. Conf.*, 2003, pp. 229–236.

[41] L. Chiaraviglio, D. Ciullo, M. Mellia, and M. Meo, "Modeling sleep mode gains in energy-aware networks," *Comput. Netw.*, vol. 57, no. 15, pp. 3051–3066, 2013.
[42] V. Eramo, E. Miucci, and M. Ammar, "Study of reconfiguration cost and energy aware VNE policies in cycle-stationary traffic scenarios," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 5, pp. 1281–1297, May 2016.
[43] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing spa violations," in *Proc. 10th IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, 2007, pp. 119–128.
[44] E. G. CoffmanJr., J. Csirik, G. Galambos, S. Martello, and D. Vigo, "Bin packing approximation algorithms: Survey and classification," in *Handbook of Combinatorial Optimization*, Eds. P. M. Pardalos, D.-Z. Du, and R. L. Graham, New York, NY, USA: Springer, 2013, pp. 455–531.
[45] *TU Grids Workload Archive - Materna 2016 Trace*. [Online]. Available: http://gwa.ewi.tudelft.nl/datasets/gwa-t-13-materna, Last Accessed on: Nov. 20, 2017.
[46] A. Kohne, M. Spohr, L. Nagel, and O. Spinczyk, "FederatedCloudSim: A SLA-aware federated cloud simulation framework," in *Proc. 2nd Int. Workshop CrossCloud Syst.*, 2014, Art. no. 3.
[47] A. Kohne, D. Pasternak, L. Nagel, and O. Spinczyk, "Evaluation of SLA-based decision strategies for VM scheduling in cloud data centers," in *Proc. 3rd Workshop CrossCloud Infrastructures Platforms*, 2016, Art. no. 6.
[48] *EnergyStar*. [Online]. Available: https://www.energystar.gov/index.cfm?c=archives.enterprise_servers, Last Accessed on: May 22, 2017.
[49] *Cisco Data Sheet Switching Infrastructure*. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-2960-x-series-switches/data_sheet_c78-728232.html, Last Accessed on: May 20, 2017.
[50] *Cisco Power Data Sheet*. [Online]. Available: http://blogs.cisco.com/enterprise/reduce-switch-power-consumption-by-up-to-80, Last Accessed on: May 20, 2017.
[51] C. Canali, R. Lancellotti, and M. Shojafar, "A computation- and network-aware energy optimization model for virtual machines allocation," in *Proc. 6th Int. Conf. Cloud Comput. Serv. Sci.*, 2017, pp. 1–11.
[52] S. Arrhenius, *Über die Reaktionsgeschwindigkeit bei der Inversion von Rohrzucker durch Säuren*. Leipzig, Germany: Wilhelm Engelmann, 1889.
[53] P. Roubaud, G. Ng, G. Henshall, R. Bulwith, R. Herber, S. Prasad, F. Carson, S. Kamath, and A. Garcia, "Impact of intermetallic growth on the mechanical strength of Pb-free BGA assemblies," *Proc. APEX*, San Diego, CA, Jan. 2001, pp. 1–13.
[54] J.-P. Clech, D. M. Noctor, J. C. Manock, G. W. Lynott, and F. Baders, "Surface mount assembly failure statistics and failure free time," in *Proc. 44th Electron. Compon. Technol. Conf.*, 1994, pp. 487–497.
[55] F. D'Andreagiovanni and G. Caire, "An unconventional clustering problem: User service profile optimization," in *Proc. IEEE Int. Symp. Inform. Theory*, Jul. 2016, pp. 855–859.
[56] H. Cambazard, D. Mehta, B. O'Sullivan, and H. Simonis, "Bin packing with linear usage costs – an application to energy management in data centres," in *Proc. 9th Int. Conf. Principles Practice Constraint Program*, 2013, pp. 47–62.
[57] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. 2nd Conf. Symp. Networked Syst. Des. Implementation*, 2005, pp. 273–286.

Luca Chiaraviglio received the master's degree in computer engineering (summa cum laude) from the Politecnico di Torino (Italy), in 2007, and the PhD degree in communications and electronic engineering from the Politecnico di Torino, Italy, in 2011. He is a tenure-track assistant professor with the Department of Electronic Engineering of the University of Rome Tor Vergata since 2016. His research interests include 5G networks, cloud data centers, Internet for rural and low income areas, and in general optimization techniques. He is a senior member of the IEEE. For additional information: https://sites.google.com/site/lucachiaraviglio/

Fabio D'Andreagiovanni received the MSc degree in industrial engineering, in 2006, and the PhD degree in operations research, in 2010 from the Sapienza Universit di Roma. He has been a 1st class research scientist with the French National Center for Scientific Research (CNRS) and at the Laboratory "Heudiasyc" of UTC - Sorbonne University since October 2016. His research has been focused on theory and applications of robust optimization and mixed integer programming. He is a member of the IEEE. For additional information: https://www.hds.utc.fr/fdandrea/index.html

Riccardo Lancellotti received the Laurea degree in computer engineering summa cum laude from the University of Modena and Reggio Emilia, in 2000, and the PhD degree in computer engineering from the University of Roma "Tor Vergata", in 2003. He has been a researcher with the University of Modena and Reggio Emilia since 2005. His research interests include geographically distributed systems, cloud computing, social networks, and peer-to-peer systems. He is a member of the IEEE Computer Society and ACM. For additional information: http://web.ing.unimo.it/rlancellotti

Mohammad Shojafar received the MSc degree in computer science from Qazvin Islamic Azad University, Iran, in 2010, and the PhD degree in information communication telecommunications from the University of Rome Sapienza, in 2016. He has been a senior researcher with the University of Padua since 2018. His research interests include 5G networks, cloud data center, green networking, computer security, and wireless sensor networks. He is a member of IEEE Computer Society. For additional information: http://mshojafar.com

Nicola Blefari-Melazzi is a full professor of telecommunications with the University of Roma Tor Vergata. His research interests include the performance evaluation, design, and control of telecommunications networks. He is a senior member of the IEEE. For additional information: http://blefari.eln.uniroma2.it/

Claudia Canali received the Laurea degree summa cum laude in computer engineering from the University of Modena and Reggio Emilia, in 2002, and the PhD degree in information technologies from the University of Parma in 2006. She has been a researcher with the Department of Engineering, University of Modena and Reggio Emilia since 2008. Her research interests include cloud computing, social networks analysis, and wireless systems for mobile Web access. She is a member of IEEE Computer Society. For additional information: http://weblab.ing.unimo.it/people/canali