Minimizing Energy Consumption of Computing-plus-Communication Tasks in Virtualized Networked Data Centers

Mohammad Shojafar*, Claudia Canali*, Riccardo Lancellotti* and Enzo Baccarelli†

*Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia,
Email:{mohammad.shojafar, claudia.canali, riccardo.lancellotti}@unimore.it
†Department of Information Engineering and Telecommunication, Sapienza University of Rome
Email: {enzo.baccarelli}@uniroma1.it

***Abstract*—In this paper, we propose a dynamic resource provisioning scheduler to maximize the application throughput and minimize the computing-plus-communication energy consumption in virtualized networked data centers. The goal is to maximize the energy-efficiency, while meeting hard QoS requirements on processing delay. The resulting optimal resource scheduler is adaptive, and jointly performs: *i)* admission control of the input traffic offered by the cloud provider; *ii)* adaptive balanced control and dispatching of the admitted traffic; *iii)* dynamic reconfiguration and consolidation of the Dynamic Voltage and Frequency Scaling (DVFS)-enabled virtual machines instantiated onto the virtualized data center. The proposed scheduler can manage changes of the workload without requiring server estimation and prediction of its future trend. Furthermore, it takes into account the most advanced mechanisms for power reduction in servers, such as DVFS and reduced power states. Performance of the proposed scheduler is numerically tested and compared against the corresponding ones of some state-of-the-art schedulers, under both synthetically generated and measured real-world workload traces. The results confirm the delay-vs.-energy good performance of the proposed scheduler.**

***Index Terms*—Virtualized Data Centers, Cloud Computing, Resource Allocation, Energy-efficiency, Lyapunov Optimization.**

I. Introduction

Over the last years, the evolution of demands for modern applications combined with the adoption of the cloud computing paradigm has led to the establishment of large-scale virtualized data centers. They are characterized by a large (and ever increasing) number of servers, which rely on hardware-assisted virtualization and multi-hop networks to provide server interconnection [1].

In a Software-as-a-Service (SaaS) scenario, two main needs must be taken into account. First, QoS requirements, typically expressed in the form of a Service Level Agreements (SLAs), must be met. Second, we also aim to reduce the energy consumption, in order to meet the demand of the so-called green computing (or, more pragmatically, we need to reduce power consumption to maximize the economic revenue). The perspective of this paper is that of an entity that is in charge of providing services to clients and controls the cloud infrastructure as well. This scenario is typical when a SaaS provider manages its own physical infrastructure, or when a company deploys its own private cloud platform.

However, current proposals are still far from solving all the problems that characterize the management of large cloud data centers. For example, several existing solutions [2]–[4] rely on the forecast of future workload and infrastructure resource demands. This may be a problem due to inherent errors in the prediction steps, that may be exacerbated by the presence of highly variable and unpredictable fluctuations of the workload patterns. Even solutions that do not rely on prediction have the critical shortcoming of considering only the resources of the servers, without considering the power consumption of the underlying intra-data center network [5]. A specular set of works focuses on the network without providing a detailed view of the computational infrastructure [6]–[8]. Only recently, the authors proposed some solutions to manage data centers considering both computational and networking infrastructure [9].

In this paper, we present a further step with respect to these proposals by introducing an innovative mechanism that combines admission control, request management through scheduling, and server management by using DVFS and reduced power states of the servers. The final goal is to manage a cloud networked data center in an energy-efficient way, while meeting QoS requirements. The main contributions of our proposal are the following ones: (1) we manage unpredictable changes of workload without the need for estimation and prediction mechanisms; (2) we consider virtual machine (VM) consolidation and task replacements; (3) we take into account both the computing (including power state changes and DVFS) and the communication energies in the management of the infrastructure.

Our proposal is tested using a simulative approach based on both real traces and synthetic workloads. The scheduler is tested under different data center scenarios. Furthermore, its performance is compared with the ones of the recent GRADient-based Iterative Scheduler (GRADIS) in [10], the Static Lyapunov-based Scheduler (SLS) in [11], the hybrid NetDC scheduler (H-NetDC) in [12], and the NetDC scheduler in [9] by considering throughput, delay, and energy costs

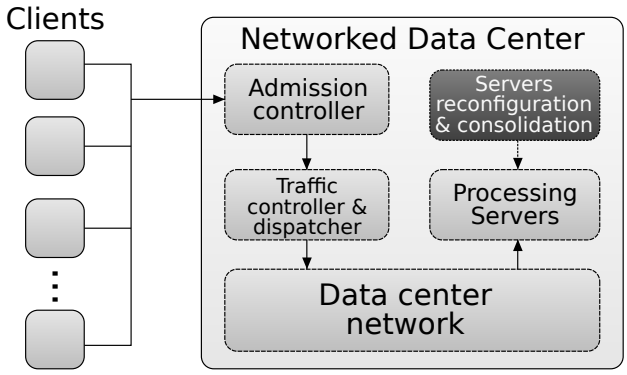


Fig. 1: Data center overview

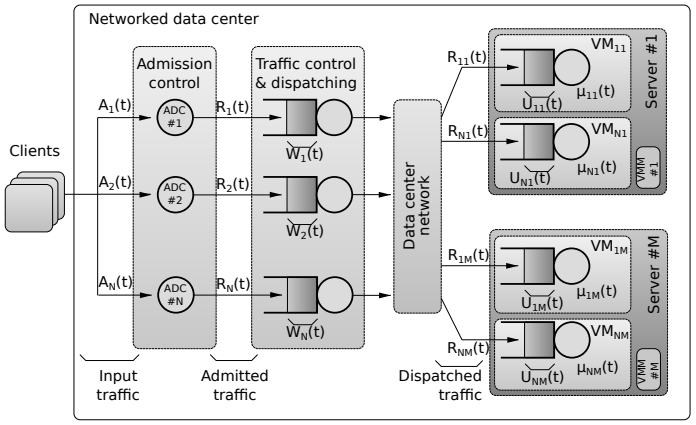


Fig. 2: Data center detailed model

(e.g., computing plus communication costs). Our experiments prove that our proposal maximizes energy efficiency, while per-application service rate is also maximized according to the control policies we define for each server.

The rest of this paper is organized as follows. After presenting the model of the considered virtualized data center architecture in Section II, we provide the problem formulation and the proposed joint scheduler in Section III. Performance evaluation results obtained through extensive simulation tests are presented in Section IV. Finally, Section V presents some concluding remarks.

II. System Architecture

A. Networked data center overview

We begin to present an overview of the considered networked data center that uses our solution for performing scheduling and admission control.

Fig. 1 provides a general scheme of the considered networked data center. We have an heterogeneous set of clients issuing requests to the networked data center (e.g., the boxes on the left side of the figure). The set of requests is first processed by the *admission controller* that decides whether or not the request should be accepted by the data center. The traffic of admitted requests is, then, handled by the *traffic controller and dispatcher*. This component dispatches the requests over the VMs hosted by the processing nodes, while buffering the requests that are not immediately sent to the VMs. Our data center supports multiple services that can be requested by the clients, and each VM may host only one service; hence, queuing and dispatching operations take into account the deployment of VMs over the server infrastructure. Requests are distributed through the *internal data center network* to the appropriate *processing servers* (and VMs). A final element of our infrastructure is the internal service of *servers reconfiguration and consolidation*. This service is in charge of the capacity adjustment, that decides the deployment of VMs over the servers and manages the servers by determining the ON/OFF status of the servers, as well as the CPU frequencies by exploiting the DVFS capabilities of the underlying hardware.

The virtualized data center shown in Fig. 2 provides a more detailed view of three main components: *ADmission Controller (ADC)*, *Dispatcher*, and *Virtual Machine Monitor (VMM)*. The proposed model is consistent with the architecture presented in [11]. We consider a separate ADC for each application. A set of dispatchers, one for each application, has the twofold task of storing the allowed requests in buffers and distributing them over the VMs that will be responsible for their processing. Each VM has a queue that is used to store requests to be processed. Furthermore, a per-server VMM is responsible for both managing the status of the server (power ON/OFF, CPU frequency that is, servers reconfiguration) and for deciding which VMs must run on each server (that is, servers consolidation).

We consider that the data center supports a Software as a Service (SaaS) cloud system, that provides a set $\mathcal{A}$ of *applications*. Let $N = |\mathcal{A}|$ be the number of such applications. The VMs that support the execution of these applications are deployed over a set $\mathcal{S}$ of M servers. For this purpose, we assume that each server $j \in \mathcal{S}$ has a set of resources (i.e., CPU, disk, memory, etc.) that are binded to the VMs hosted on it and supervised by its *VMM*. In practice, this *VMM* resides on the host Operating System of each virtualized server. In this paper, we focus on the case where the CPU is the bottleneck resource of each server. This happens, for example, when all applications running on the servers are computationally intensive. Therefore, CPU frequency, switch frequency and power and/or energy constraints are critical for this scenario [13].

According to [11], we consider a discrete-time model based on timeslots. Decisions on the capacity adjustment of the data centers are taken at the beginning of each timeslot. We assume that each timeslot is significantly longer than the job average interarrival time, so that resource provisioning may be based on the average arrival rate during each slot. For example, if requests arrive at a rate of few requests per seconds, the timeslot can be in the order of minutes.

We now detail the formal model used to describe the system. For the convenience of the readers, the major notations used in this paper are listed in Table I. In each slot t, new requests for i-th application arrive according to a random arrival process

TABLE I: Notation.

Symbol	Meaning/Role
a_{ij}	Indicator of the i-th application to the j-th server
$R_i(t)$ $(IU/slot)$	Admitted arrival requests after $ADC(i)$ in slot t
$A_i(t)$ $(IU/slot)$	New arrival requests for i-th application in slot t
$R_{ij}(t)$ $(IU/slot)$	Routing decisions of i-th application to j-th server
$\mu_{ij}(t)$ $(IU/slot)$	Service rate of i-th application on j-th server in slot t
$W_i(t)$ $(IU/slot)$	i-th application buffer size in slot t
$U_{ij}(t)$ $(IU/slot)$	i-th application buffer size of the j-th server in slot t
$\overline{RTT_j}$ (s)	Average round-trip-time of j-th dispatcher-server link
$f_j(t)$ $(IU/slot)$	Processing rate of j-th server in slot t
$f_j^{\max}(IU/slot)$	Maximum allowed processing rate of j-th server
$r_j(t)(IU/slot)$	Communication rate of the j-th server-dispatcher link
$\mathcal{E}_c^{idle}(j)(Joule)$	Static energy consumed by j-th server
$P_N^{idle}(j)(Watt)$	Static power consumed by j-th dispatcher-server link
$\mathcal{E}_c^{max}(j)(Joule)$	Maximum energy consumed by j-th server
$\mathcal{E}_c(j,t)(Joule)$	Computing energy consumed by j-th server in slot t
$\mathcal{E}_{dyn}(j,t)(Joule)$	Switch energy consumed by j-th server in slot t
$\mathcal{E}_{LAN}(j,t)(Joule)$	Communication energy consumed by j-th server
$\mathcal{E}_j^{tot}(t)(Joule)$	Total energy consumed by j-th server in slot t

$A_i(t)$ $(IU/slot)$ with time average rate λ_i $(IU/slot)$[1]. We assume that the statistics of $A_i(t)$ are unknown, so that $\{A_i(t) \in \mathcal{R}_0^+, t \geq 0\}$ is a random process with unknown statistics.

B. Admission Controller

For each application $i \in \mathcal{A}$, the ADC of Fig. 2 decides whether to admit or decline the new requests. The admitted requests are stored in the *Dispatcher* buffers (see the buffers connected to each ADC), before being routed by the *Dispatcher* to one of the servers hosting that application.

We consider (time-slotted) $G/G/1$ fluid systems for modeling the queue sets of ADC and servers of Fig. 2. Due to the admission control, both set of queues are loss-free and they implement the FIFO service discipline. Specifically, at the end of slot t, the arrival $A_i(t), t \geq 0$ of processing new (possibly, heterogeneous) requests arrive at the input of the ADCs of Fig. 2. Note that $A_i(t)$ is assumed to be independent from the current backlogs of the ADC queues of Fig. 2. We assume that any new arrival that is not admitted by the ADC of Fig. 2 is declined. Let $R_i(t)$ be the number of requests out of $A_i(t)$ that are admitted into the i-th application buffer of the Dispatcher by $ADC(i)$. Let $W_i(t) \in \mathcal{R}_0^+$ and $U_{ij}(t) \in \mathcal{R}_0^+$ be the numbers of IUs stored by ADCs and server queues of Fig. 2 *at the beginning* of slot t. Furthermore, let R_{ij} $(IU/slot)$ be the size of task of the i-th application served by the j-th server that: (i) is drained from the $ADC(i)$ queue *at the beginning* of slot t; and, (ii) is injected into the ij-th queue at *the end* of slot t. We assume that dispatching occurs on the basis of the VM-to-Server mapping provided by $\{a_{ij}\}$. If a given application is served by multiple VMs, we assume that the dispatcher uses the round-robin discipline for the scheduling.

C. Adaptive Balancing Control and Dispatching

Let $\mu_{ij}(t)$ $(IU/slot)$ be the traffic that is drained from the j-th server queue of Fig. 2 *during* slot t. We denote the buffer backlog by $W_i(t)$, with $0 \leq W_i(t) \leq A_i(t)$. Hence, the time evolutions of the backlogs $\{W_i(t) \in (\mathcal{R})_0^+, t \geq 0\}$, $\{U_{ij}(t) \in (\mathcal{R})_0^+, t \geq 0\}$ of the $ADC(i)$ and j-th server queues are dictated by the following Lindley's equations, respectively:

$$W_i(t+1) = \left[W_i(t) - \left(\sum_j a_{ij} R_{ij}(t) \right) \right]_+ + R_i(t), t \geq 0, \quad (1)$$

$$U_{ij}(t+1) = [U_{ij}(t) - \mu_{ij}(t)]_+ + R_{ij}(t), t \geq 0, \quad (2)$$

where $R_{ij}(t)$ is the number of requests for i-th application that are routed from its dispatcher to the j-th server in slot t.

D. Computing and Switching

The client requests are sent by the dispatchers to the active VMs running on the servers for being processed. In this paper, we assume that VMs deployed over a server can change their share of server resources according to the model described in [14], that is widely adopted in private cloud environments. This model tends to face conditions of high computational demand by means of few large VMs instead of many small VMs. We adopt a simplified model where a single VM is deployed over each server and uses all the available resources for that server (we will refer to the VM deployed on server j as $VM(j)$). The extension of the model to the case where multiple VMs share the same server is left as an open issue to be addressed in future works.

In this paper, we model each server as capable of a processing rate $f_j(t)$ in slot t. Depending on the size $R_{ij}(t)$ $(IU/slot)$ of the i-th application's task to be currently processed by the j-th server in slot t, the corresponding processing rate $f_j(t)$ may be adaptively scaled at run-time through DVFS. In this way, each server can be operated at multiple voltages which operate different CPU frequency [15]. Let Q be the number of allowed CPU rates: for example, the Intel EIST-technologies allows to choose the clock frequency from a set of $Q = 6$[2] or $Q = 15$[3] values for reducing the power consumption when the computational demands are low.

DVFS is applied by the hosting physical servers to stretch the processing times of the tasks and reduce energy consumption by decreasing the processing rates of the active VMs. Each processing rate may assume values over interval $[0, f_j^{\max}]$, where $f_j^{\max}$ $(IU/slot)$ is the maximum allowed processing of $VM(j)$.

In [16], it is shown that there is a quadratic relationship between the *CPU utilization* of $VM(j)$ and the corresponding computing energy consumption $\mathcal{E}_c(j,t)$ (all servers in our model are assumed to have identical CPU resources), so that we can write:

$$\mathcal{E}_c(j,t) = \mathcal{E}_c^{idle}(j) + (f_j(t)/f_j^{max})^2 \left(\mathcal{E}_c^{max}(j) - \mathcal{E}_c^{idle}(j) \right) \quad (3)$$

Furthermore, we note that switching from the processing rate $f_j(t-1)$ (e.g., the processing rate of $VM(j)$ at $(t-1)$-th slot) to $f_j(t)$ (e.g., the processing rate of $VM(j)$ in t-th slot) entails an energy overhead of $\mathcal{E}_{dyn}(j,t)$ [16]. Although the actual

[1]The meaning of an Information Unit (IU) is application-dependent. We anticipate that, in the carried out tests of Section IV, IUs are understood as $Mbit$.

[2]http://download.intel.com/design/network/papers/30117401.pdf

[3]https://software.intel.com/sites/default/files/ftalat.pdf

behavior of the function $\mathcal{E}_{dyn}(j,t)$ depends on the adopted DVFS technique, a quite common model is the following quadratic one [16]:

$$\mathcal{E}_{dyn}(j,t) = k_e \left(f_j(t) - f_j(t-1)\right)^2. \quad (4)$$

In Eq. (4), k_e $(Joule/(Hz)^2)$ denotes the switch cost induced by an unit-size rate switching, which is typically limited up to few hundreds of $\mu J's$ per $(MHz)^2$ [16].

At each slot, the VMM allocates the resources of each server among the VMs that host the applications running on that server. Let $\grave{\mathcal{S}}(t)$ be the set of turned ON VMs at slot t. Hence, we have $\grave{\mathcal{S}}(t) \subseteq \mathcal{S}$. In our reference data center, the scheduler interleaves *reconfiguration* and *consolidation* slots. Specifically, at each reconfiguration slot, the scheduler does not change the set $\grave{\mathcal{S}}(t)$ of turned ON VMs and, then, it leaves unchanged the corresponding sets of turned ON/OFF physical servers. However, it assigns the requests R_{ij} to the turned ON VMs. Thus, the routing decisions $R_{ij}(t)$ must satisfy the following constraints at slot t:

$$a_{ij} = \begin{cases} 1, & R_{ij} = 0, \text{if } j \notin \grave{\mathcal{S}}(t) \\ 0, & \text{if } j \in \grave{\mathcal{S}}(t); \end{cases} \quad (5)$$

$$0 \leq \sum_{j \in \grave{\mathcal{S}}(t)} a_{ij} R_{ij}(t) \leq W_i(t) \quad (6)$$

E. Communication

We assume that the j-th virtual end-to-end connection (e.g., the j-th virtual link) of the data center of Fig. 2 is bidirectional, symmetric and operates in a *half-duplex* way [17]. A joint analysis of the computing-plus-communication energy consumption in *delay-tolerant* data centers is performed in [16], where the effects of inter networking infrastructures are evaluated. Two main conclusions arise from [16]. First, the energy consumption due to the data transport may represent a large part of the total energy consumption, especially at medium/high usage rates. Second, the energy consumption of cloud infrastructures needs to be analyzed by *simultaneously* accounting for data computing and Dispatcher-Server data transport. Motivated by these considerations, we consider the TCP New Reno protocol [18] to model the managed end-to-end Dispatcher-Server transport connections. Hence, under the Congestion Avoidance state, the power drained by each connection may be evaluated as in [9]:

$$P_j^{net}(t) = \Omega_j \left(\overline{RTT_j}\ r_j(t)\right)^2 + P_N^{idle}(j),\ j = 1, \ldots, M, \quad (7)$$

where $\overline{RTT_j}$ is the average round-trip-time of the j-th server-dispatcher end-to-end connection, $r_j(t)$ is the communication rate of the j-th virtual link at the t-th slot, and Ω_j is the j-th virtual link power coefficient that depends on the maximum segment size of the transmitted packets and on the number of per-segment ACKs as described in [10], [19].

Hence, the corresponding one-way transmission delay is: $D_j(t) = (R_{ij}(t))/r_j(t)$, where $R_{ij}(t)$ is the workload of the i-th application (requests) which is assigned to available j-th server at the t-th slot. The overall two-way computing-plus-communication delay induced by the j-th end-to-end connection of Fig. 2 equates to $2D_j$, so that the hard constraint on the overall per-request execution time is: $\max_{1 \leq j \leq M}\{2D_j\}$. Thus, the corresponding one-way communication energy $\mathcal{E}_{LAN}(j,t)$ wasted by the j-th virtual link at slot t is:

$$\mathcal{E}_{LAN}(j,t) = P_j^{net}(t)\left(a_{ij} R_{ij}(t)\right)/r_j(t), \forall i \in \mathcal{A}, \quad (8)$$

Hence, the resulting total computing-plus-communication energy $\mathcal{E}_j^{tot}(t)$ consumed at slot t by the j-th server is:

$$\mathcal{E}_j^{tot}(t) = \mathcal{E}_c(j,t) + \mathcal{E}_{dyn}(j,t) + \mathcal{E}_{LAN}(j,t). \quad (9)$$

III. Problem Formulation and Solution

We now present our proposal of a dynamic load balancing and resource provisioning approach that takes into account computation and communication costs, and exploits DVFS-based discrete frequencies and their time shares for each VM. Our final goal is to maximize the throughput of the applications and minimize the joint computing-plus-communication energy costs of the servers, subject to the available control options and to the structural constraints imposed by the considered model. Since frequent turning ON/OFF servers may be undesirable (for example, due to hardware reliability issues), we will focus on frame-based control policies, where time is divided into frames of length T slots. We define T_{max} as the maximum number of slots considered in our experiments for the evaluation of SLAs. We recall that the set of active servers is chosen at the beginning of each frame and does not change for the duration of that frame (reconfiguration state): this set may change in the next frame as workloads change (consolidation state). In the consolidation state, we use the MBFD [20] policy to migrate some of active servers' backlogs to some other servers, then we update the set of active servers. Let $\mathcal{T}_i$ and ξ_j denote the (average expected) rate of admitted requests for i-th application and the (average expected) total joint computing-plus-communication energy consumption of j-th server, that is $\mathcal{T}_i \triangleq \lim_{t \leftarrow \infty} \frac{1}{t-1} \sum_{\tau=0}^{t-1} E\{R_i(\tau)\}$ and $\xi_j \triangleq \lim_{t \leftarrow \infty} \frac{1}{t-1} \sum_{\tau=0}^{t-1} E\{\mathcal{E}_j^{tot}(\tau)\}$, respectively. We define $\underline{\mathcal{T}} \triangleq \{\ \mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_N\ \}$ as the vector of average rates of applications and $\underline{\xi} \triangleq \{\ \xi_1, \xi_2, \ldots, \xi_M\ \}$ as the vector of energy average rates of servers. Hence, the resource reconfiguration/consolidation problem at slot t is formulated as in the following:

$$\min_{\chi} \quad \left(\theta \sum_{j \in \grave{\mathcal{S}}} \xi_j - \sum_{i \in \mathcal{A}} \beta_i \mathcal{T}_i\right), \quad (10.1)$$

subject to:

$$0 \leq f_j(t) \leq f_j^{max}, \qquad \forall j \in \grave{\mathcal{S}}, \quad (10.2)$$

$$0 \leq \mathcal{T}_i \leq \lambda_i, \underline{\mathcal{T}} \in \Gamma, \qquad \forall i \in \mathcal{A}, \quad (10.3)$$

$$\text{Eqs. } (1), (2), (5), (6). \quad (10.4)$$

where $\chi \triangleq \{\underline{\mathcal{T}}, f_j(t), U_{ij}, R_i(t), i \in \mathcal{A}, j \in \grave{\mathcal{S}}\}$. In Eq.(10.1), θ and β_i are non-negative weights which are used for normalization and to assign priorities between throughput and energy. The optimization problem in Eq. (10.1) is a general weighted linear combination of the sum throughput of the applications and the average energy usage in the data center. Eq. (10.2)

(resp., Eq. (10.3)) bounds the maximum processing rate of j-th server (resp., maximum average rate of i-th application). At the end, Γ in (10.3) represents the *capacity region* of the data center, which is defined as the set of all possible longterm throughput values that can be achieved under any feasible resource allocation strategy.

In the rest of the paper, the optimal solution of the problem (10) is denoted as in $\{ f_j^*(t), U_{ij}^*, R_i^*(t), i \in \mathcal{A}, j \in \mathcal{S} \}$. We rely on the Lyapunov optimization [11], [21] to develop an optimal control policy. In particular, we introduce a dynamic control algorithm that achieves the optimal solution $\{\mathcal{T}_i^*\}$ and $\{\xi_j^*\}$. We use queue backlog values in slot t to make decisions for the proposed algorithm in ADC, Routing and VMM components of Fig. 2. These decisions help us to dynamically update the system condition, while the new requests come to the system. Hence, the proposed algorithm optimizes the objective in (10) by solving a sequence of optimization problems over time. The queue backlogs can be viewed as dynamic Lagrange multipliers that enable stochastic optimization [21]. The proposed method called Joint Dynamic Lyapunov-based Scheduler *(JDLS)* performs the following actions:

1) *Admission control*: it is important to maximize $R_i(t)$ and minimize the application backlog $W_i(t)$ for i-th application in slot t. So, we should solve the following problem:

$$\min_i \; R_i(t)\,[W_i(t) - V\beta_i]\,, \tag{11.1}$$

$$\text{s.t: } 0 \leq R_i(t) \leq A_i(t), \tag{11.2}$$

where $V \geq 0$ is a control parameter used by the system administrator to control the trade-off between average delay and total average utility by exploiting the threshold-based mechanism used in [11]. If the current ADC queue backlog for i-th application satisfies the condition $W_i(t) > V\beta_i$, then $R_i^*(t) = 0$ and no new requests are admitted. Otherwise, $R_i^*(t) = A_i(t)$ and all new requests are admitted.

2) *Request dispatching*: for the i-th application, we dispatch the $ADC(i)$'s drained queue requests to the active servers in set $\grave{\mathcal{S}}(t)$ using a *round-robin policy*.

3) *VMM and resource allocation*: for j-th server in the set $\grave{\mathcal{S}}(t)$, we have to solve the following sub-problem:

$$\min_i \; \sum_{i \in \mathcal{A}} V\theta\mathcal{E}_j^{tot}(t) - U_{ij}(t)\mu_{ij}(t), \tag{12.1}$$

$$\text{s.t: } Eqs.\,(3), (4), (7), \tag{12.2}$$

Eq. (12) is a generalized min-weight problem, where the service rate provided to any application is weighted by its current queue backlog. Hence, the optimal solution allocates resources to maximize the service rate of the most backlogged application: $\sum_{i \in \mathcal{A}} U_{ij}(t)\mu_{ij}(t)$, and minimize the overall energy consumption for j-th server: $\sum_{i \in \mathcal{A}} V\theta\mathcal{E}_j^{tot}(t)$. It is worth to note that each server solves its own resource allocation problem independently by using the queue backlog values of the applications hosted on it and this can be implemented in a fully distributed way, with a major benefit in terms of scalability.

Algorithm 1 reports a pseudo-code for the adaptive implementation of the proposed resource scheduler.

Algorithm 1 A pseudo-code of the proposed JDLS

```
1: for t ≥ 1 do
2:     if t ≠ nT, then                          ▷ t is a reconfiguration slot,
3:         for all i ∈ A do
4:             Update R_i(t) using eq. (11);
5:             Update R_ij(t) using eq. (5), (6) and JDLS's Routing;
6:             Update W_i(t+1) using eq. (1);
7:         end for
8:         for all j ∈ Ŝ do
9:             Update μ_ij(t) using eq. (12);
10:            Update U_ij(t+1) using eq. (2);
11:        end for
12:    else                                     ▷ t is a consolidation slot
13:        re-run lines 3-7;
14:        for all j ∈ Ŝ do
15:            Update μ_ij(t) using eq. (12) and update eq. (3), (4),
    and (8) with zero frequency;
16:            Request task replacement using MBFD [20] policy
    over Ŝ(t);
17:            Update Ŝ(t);
18:        end for
19:    end if
20: end for
```

IV. Test Results and Performance Comparisons

This section presents the performance evaluation of the proposed scheduler under a set of synthetic and real-world input traffic traces. In particular, we perform a sensitivity analysis with respect to the main model parameters; then, we compare the performance of the proposed JDLS scheduler with that of the recent GRADient-based Iterative Scheduler (GRADIS) [10], the Static Lyapunov-based Scheduler (SLS) in [11], the hybrid NetDC scheduler (H-NetDC) in [12], and the NetDC scheduler (NetDC) in [9]. It is worth to note that the parameters of each scheduler have been tuned in preliminary experiments to ensure a fair comparison among the considered approaches. As metrics for the scheduler performance evaluation, we consider the total utility (Throughput) $\mathcal{U}_{tot}$, the total consumed energy $\overline{\xi}_j^*$ and the total delay $\overline{T}_{tot}^*$ of the admitted requests.

A. Simulated Setup

The simulations have been carried out by exploiting the numerical software of the MATLAB platform. They emulate 10 quad-core Dell PowerEdge servers, equipped with 3.06 GHz Intel Xeon CPU and 4GB of RAM. All the emulated servers are connected through commodity Fast Ethernet NICs. In all carried out tests, we configure the VMs with 512MB of RAM and emulate the TCP New Reno protocol for implementing the needed VM-to-VM transport connections.

For the experiments, we consider two different scenarios. The first scenario includes a synthetic workload, whose main parameters are detailed in Table II. The second scenario refers

TABLE II: Default values of the main simulated parameters.

Parameter	Value
$(\Delta,\ \gamma = \theta/\beta_i)$	(1 (s), {0.5, 1, 2})
$(T,\ T_{max})$	(100, 1000)
$(\Omega_j,\ P_N^{idle}(j))$	(0.5, 0.5) $(Watt)$ $\forall j \in \mathcal{S}$
$(M,\ N,\ V)$	(100, 10,{0:500:20000})
A_i	8 $(Mbit)$ $\forall i \in \mathcal{A}$
PMR[4]	2
k_e, Q	0.05 $(J/(MHz)^2)$, 6
r_j^{max}, f_j^{max}	100, 10 $(Mbit/s)$ $\forall j \in \mathcal{S}$
$\overline{RTT_j}$	70 (μs) $\forall j \in \mathcal{S}$
$(\mu_{ij}^{min},\ \mu_{ij}^{max})$	{(200, 400), (50, 300)} $(Mbit/slot)$
$(\mathcal{E}_c^{idle}(j),\ \mathcal{E}_c^{max}(j))$	(5, 240)$(Joule)$ $\forall j \in \mathcal{S}$

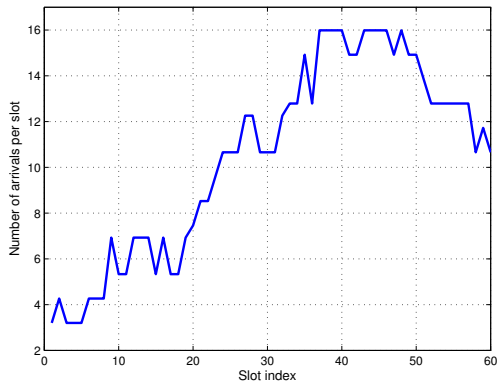
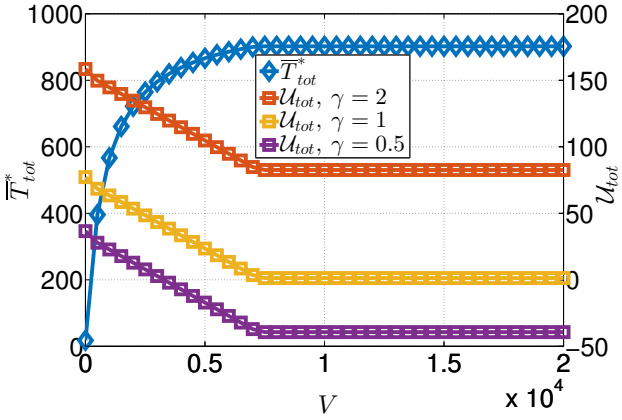
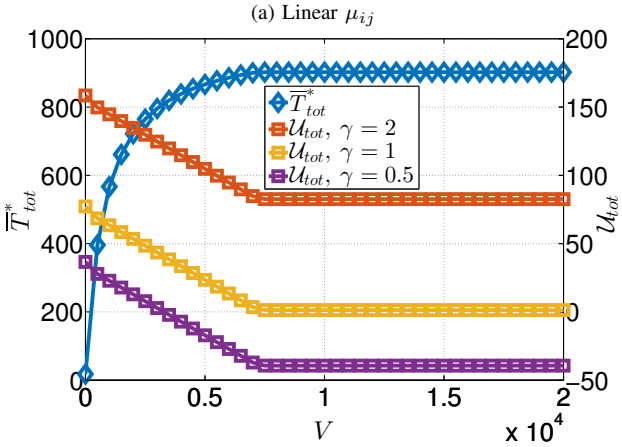


Fig. 3: Measured workload trace: PMR = 1.526.



(a) Linear μ_{ij}



(b) Quadratic μ_{ij}

Fig. 4: $\overline{T}^*_{tot}$ vs. V and $\mathcal{U}_{tot}$ for the synthetic workload for linear (4a) and quadratic (4b) control decision μ_{ij}.

to a real-world workload trace, which is represented in Fig. 3: this is the same real-world workload trace considered in [22]. We perform preliminary experiments and we found that the best parameter values for the real-world workload are $k_e = 0.5$ $[Joule/(MHz)^2]$ and $T = 1.2$ $[s]$.

B. Experimental Results

In order to measure the performance of the proposed scheduler JDLS, we perform different experiments detailed in the following subsections.

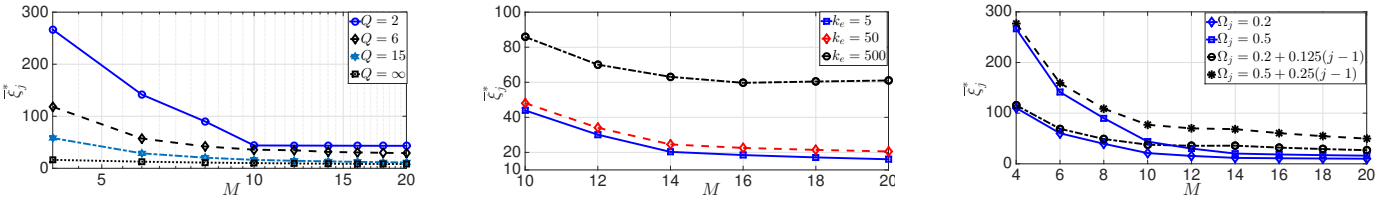
1) Throughput and Average Total Delay: In this experiment we evaluate the total average utility $\mathcal{U}_{tot}$ and the average total delay of the admitted requests $\overline{T}^*_{tot}$ for different values of the input parameters V and γ. We define $\overline{T}^*_{tot}$ like [23] as the sum of the delay in the first queue blocks (ADC buffers) and second queue blocks (VM buffers), expressed as the number of time slots. Fig. 4a and 4b show the measured $\mathcal{U}_{tot}$ and $\overline{T}^*_{tot}$ under the linear and quadratic control decisions μ_{ij} of Eq. (12), respectively.

The observation of these results leads to two main conclusions. First, $\mathcal{U}_{tot}$ (which is the solution of (10.1)) decreases for increasing values of the weight factor of $\gamma \triangleq \theta/\beta_i$, $\forall i$ and V. $\mathcal{U}_{tot}$ converges to a minimum value, and $\overline{T}^*_{tot}$ increases and converges to a maximum value for larger values of V. This is due to the fact that, according to the conditional Lyapunov optimization approach in [24] (exploiting Lyapunov drift theorem), the performance of the minimization algorithm is bounded up to a finite constant which depends on the arrival and service rates [24]. Second, we conclude that the average overall time and utility of the presented data center is independent of the control decision μ_{ij}. It means that each server solves its own resource allocation problem independently by using the queue backlog values of the applications hosted on it and this allows, in return, the implementation of the proposed approach in a fully distributed way.

2) Sensitivity to the data center parameters: Fig. 5 reports the average total consumed energy $\overline{\xi}^*_j$ for servers for various values of M and other parameters, such as Q (Fig. 5a), k_e (Fig. 5b), and Ω_j (Fig. 5c). Specifically, Fig. 5 points out that: (i) while M increases, the $\overline{\xi}^*_j$ decreases; (ii) while Q, k_e, and Ω_j increases, the $\overline{\xi}^*_j$ declines according to the Eqs. (3), (4) and (8), respectively. In Fig. 5a, $Q = \infty$ indicates the case of real-time results where we have no queues: in this case, the optimal frequency of each server is the corresponding frequency that we need for the incoming workload (ideal case).

3) Performance Comparisons: We now present a comparison of the proposed JDLS scheduler with other available

[4]PMR:=Peak-to-Mean Ratio of the input workload



(a) Q (b) k_e (c) Ω_j

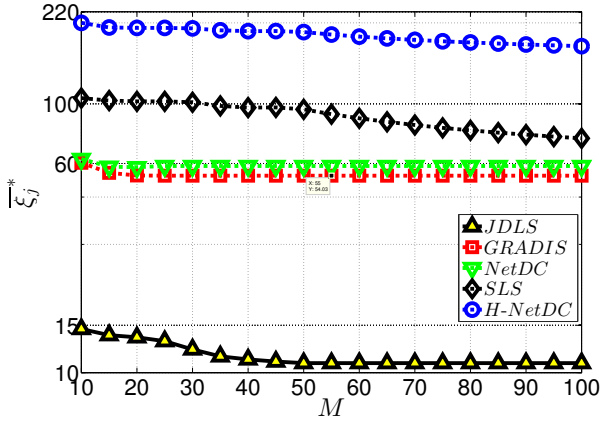
Fig. 5: Sensitivity analysis with respect to parameters Q (5a), k_e (5b) and Ω_j at $V = 10^4$, $k_e = 5$ (5c).

alternatives in terms of consumed energy, total delay and execution times.

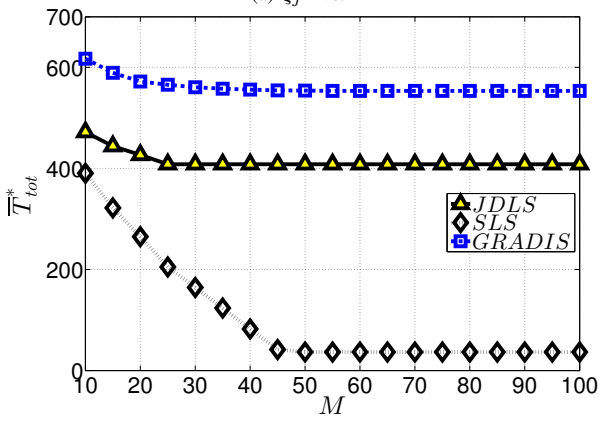
Fig. 6 presents the average total consumed energy $\overline{\xi}_j^*$ and the average total delay $\overline{T}_{tot}^*$ versus the number of VMs M for the aforementioned schedulers [9]–[12] for the synthetic workload. In detail, Fig. 6a shows that the average energy saving of the JDLS scheduler over NetDC, GRADIS, SLS H-NetDC methods is about 75%, 70%, 85% and 91%, respectively. This result confirms that the proposed JDLS scheduler is capable to adapt to the incoming behaviors of the input workload by increasing the number of turned ON VMs faster than other methods. Fig. 6b compares the average total delay $\overline{T}_{tot}^*$ of JDLS, GRADIS [10] and SLS [11] for increasing values of M. In this case we do not consider the NetDC and H-NetDC schedulers because they are real-time approaches which do not use any queue for the incoming requests. The results of Fig. 6b show that the average total saving of JDLS over the GRADIS is almost 30%, but is lower with respect to SLS; it is important to consider that JDLS performs communication joint switching optimization whilst SLS does not perform them.

We also evaluate how JDLS can handle a real world scenario. To this aim, we consider the traces in Fig. 3 [22] previously described. Fig. 7 compares the average total consumed energy $\overline{\xi}_j^*$ for different values of M for the aforementioned schedulers [9]–[12]. These results show that by increasing the number of servers, the average energy consumption per-server decreases for all the tested schedulers. The amount of energy saving of JDLS compared to SLS, NetDC, GRADIS, and H-NetDC is about 30%, 50%, 51% and 78%, respectively. This last comparison confirms that JDLS can reduce energy consumption even in the highly variable scenario that characterizes real world workloads.

In the last simulation, we evaluate the average execution time for each of the five considered schedulers for different number of slots ($T_{max} = 10^3$, 10^4) for the synthetic and the real-world workload traces. The results reported in Table III show that the JDLS scheduler is able to work with a significantly lower average execution time with respect to the GRADIS and the NetDC/H-NetDC methods, in both scenarios and scales of incoming workloads. The NetDC/H-NetDC methods are considered together because, from a computational point of view, they perform the same operations and therefore have the same speed. As shown in Table III, the average time saving of the JDLS scheduler for the higher amount of time slots ($T_{max} = 10^4$) over the GRADIS and



(a) $\overline{\xi_j}^*$-vs.-M



(b) $\overline{T}_{tot}^*$-vs.-M

Fig. 6: Performance comparison under synthetic workload (with $T_{max} = 10^4$ and $V = 10^4$).

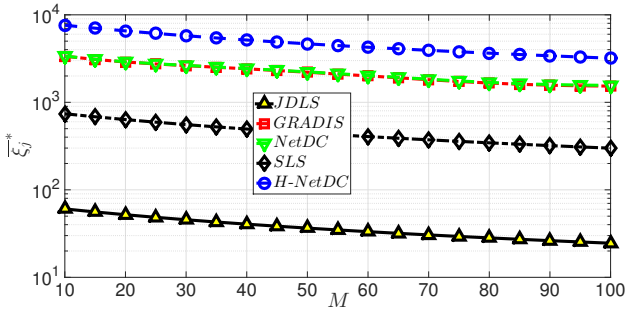


Fig. 7: $\overline{T}_{tot}^*$ comparison under real workload.

the NetDC/H-NetDC alternatives is of about 48% and 94.7% in the synthetic workload, and of 59% and 66% in the real-

TABLE III: Average execution time (ms) of considered schedulers for synthetic and real-world workloads.

Workload	JDLS	SLS	GRADIS	NetDC/ H-NetDC
Real-world ($T_{max} = 10^4$)	26.7	25.8	65.3	78.6
Real-world ($T_{max} = 10^3$)	3.6	2.9	7	7.3
Synthetic ($T_{max} = 10^4$)	44.1	38.6	85.1	840
Synthetic ($T_{max} = 10^3$)	4	3.4	8	84.8

world workload, respectively. On the other hand, with respect to the SLS scheduler, the JDLS execution time is about 3% (0.9 ms) and 13% (5.5 ms) higher in the same scale of incoming workload ($T_{max} = 10^4$). It is important to note that, differently from the SLS scheduler, the JDLS proposal includes the communication and switch components in the model: even with this additional element, the execution time of JDLS is comparable with the SLS alternative.

To summarize our results, we can conclude that the JDLS scheduler can significantly outperform every other considered solution in reducing the computing-plus-communication energy for a wide range of workloads (including real-world traces) and parameter setups. Furthermore, the scheduler is extremely fast in taking decisions, suggesting that it has potential for scalability over large data centers and large input workloads.

V. Conclusions and Future Work

In this paper, we proposed an effective dynamic scheduler for the joint adaptive tuning of the: (i) admitted traffic; (ii) delivered throughput; and (iii) resource reconfiguration and consolidation of virtualized networked data center platforms. The overall goal is to reduce the energy consumption while guaranteeing QoS in cloud scenarios characterized by high computational demand and delay sensitivity. Remarkable features of the developed joint scheduler are that: (i) its implementation is distributed and adaptive, and the resulting complexity scales with the number of the available VMs; (ii) it minimizes the energy consumed by the overall platform for computing, router-server communication and server reconfiguration; and, (iii) despite the unpredictable time-varying nature of the input workload, it is capable to provide QoS guarantees, in terms of maximizing delivered average throughput, and *maximum* queuing-plus-computing delay. Actual performance of the proposed scheduler has been numerically tested under both synthetic and real-world traces for the input traffic under various conditions, allowing us to drive analytical performance guarantees of the algorithm. This work can be extended in some directions of potential interest. In particular, we can enrich the application scenario to consider intra-slot traffic arrivals and we can introduce live migration of VMs to achieve additional energy savings.

VI. Acknowledgement

The first three authors acknowledge the support of the University of Modena and Reggio Emilia through the project *SAMMClouds: Secure and Adaptive Management of Multi-Clouds*.

References

[1] J. Whitney and P. Delforge, "Data center efficiency assessment – scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers," NRDC, Anthesis, Tech. Rep., 2014.

[2] D. Ardagna, M. Ciavotta, and R. Lancellotti, "A receding horizon approach for the runtime management of iaas cloud systems," in *16th IEEE SYNASC*, 2014, pp. 445–452.

[3] C. Canali and R. Lancellotti, "Exploiting classes of virtual machines for scalable iaas cloud management," in *Proc. of the 4th NCCA*, 2015, pp. 15–22.

[4] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia, "A survey on virtual machine migration and server consolidation frameworks for cloud data centers," *Journal of Network and Computer Applications*, vol. 52, pp. 11–25, 2015.

[5] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang, "Energy-aware autonomic resource allocation in multitier virtualized environments," *Services Computing, IEEE Transactions on*, vol. 5, no. 1, pp. 2–19, Jan 2012.

[6] L. Chiaraviglio, D. Ciullo, M. Mellia, and M. Meo, "Modeling sleep mode gains in energy-aware networks," *Computer Networks*, vol. 57, no. 15, pp. 3051–3066, 2013.

[7] N. Cordeschi, T. Patriarca, and E. Baccarelli, "Stochastic traffic engineering for real-time applications over wireless networks," *Journal of Network and Computer Applications*, vol. 35, no. 2, pp. 681–694, 2012.

[8] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *ACM SIGCOMM*, vol. 41, no. 4, 2011, pp. 50–61.

[9] N. Cordeschi, M. Shojafar, and E. Baccarelli, "Energy-saving self-configuring networked data centers," *Computer Networks*, vol. 57, no. 17, pp. 3479–3491, 2013.

[10] M. Shojafar, C. Nicola, and E. Baccarelli, "Energy-efficient adaptive resource management for real-time vehicular cloud services," *IEEE Transactions on Cloud Computing (TCC)*, vol. PP, no. 99, p. 1, 2016.

[11] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely, "Dynamic resource allocation and power management in virtualized data centers," in *Proc. of IEEE/IFIP NOMS*, 2010, pp. 479–486.

[12] N. Cordeschi, D. Amendola, F. De Rango, and E. Baccarelli, "Networking-computing resource allocation for hard real-time green cloud applications," in *Wireless Days*, 2014, pp. 1–4.

[13] A. Mishra, R. Jain, and A. Durresi, "Cloud computing: networking and communication challenges," *IEEE Communications Magazine*, vol. 50, no. 9, pp. 24–25, 2012.

[14] D. Gmach, J. Rolia, and L. Cherkasova, "Selling t-shirts and time shares in the cloud," in *Proc. of 12th IEEE/ACM CCGrid*, 2012, pp. 539–546.

[15] S. Azodolmolky, P. Wieder, and R. Yahyapour, "Cloud computing networking: challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 54–62, 2013.

[16] M. Portnoy, *Virtualization essentials*. John Wiley & Sons, 2012.

[17] J. Baliga, R. W. Ayre, K. Hinton, and R. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proceedings of the, IEEE*, vol. 99, no. 1, pp. 149–167, 2011.

[18] O. Tamm, C. Hermsmeyer, and A. M. Rush, "Eco-sustainable system and network architectures for future transport networks," *Bell Labs Technical Journal*, vol. 14, no. 4, pp. 311–327, 2010.

[19] B. Kantarci and H. T. Mouftah, "Delay-constrained admission and bandwidth allocation for long-reach epon," *Journal of Networks*, vol. 7, no. 5, pp. 812–820, 2012.

[20] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[21] L. Georgiadis, M. J. Neely, and L. Tassiulas, *Resource allocation and cross-layer control in wireless networks*. Now Publishers Inc, 2006.

[22] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "Analytic modeling of multitier internet applications," *ACM Transactions on the Web (TWEB)*, vol. 1, no. 1, p. 2, 2007.

[23] M. Dabbagh, N. Sayegh, A. Kayssi, I. Elhajj, and A. Chehab, "Fast dynamic internet mapping," *Future Generation Computer Systems*, vol. 39, pp. 55–66, 2014.

[24] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.