Towards Sustainable Edge Computing through Renewable Energy Resources and Online, Distributed and Predictive Scheduling

Giovanni Perin*, *Student Member, IEEE,* Michele Berno, Tomaso Erseghe, and Michele Rossi, *Senior Member, IEEE*

***Abstract*—In this work, we tackle the energy consumption problem of edge computing technology looking at two key aspects: (i) reducing the energy burden of modern edge computing facilities to the power grid and (ii) distributing the user-generated computing load within the edge while meeting computing deadlines and achieving network level benefits (server load balancing *vs* consolidation and reduction of transmission costs). In the considered setup, edge servers are co-located with the base stations of a mobile network. Renewable energy sources are available to power base stations and servers, and users generate workload that is to be processed within certain deadlines. We propose a predictive, online and distributed algorithm for the scheduling of computing jobs that attains objectives (i) and (ii). The algorithm achieves fast convergence, leading to an energy efficient use of edge computing facilities, and obtains in the best case a reduction of 50% in the amount of renewable energy that is sold to the power grid by heuristic policies and that is, in turn, used at the network edge for processing.**

***Index Terms*—Edge computing, green computing, renewable energy, Model Predictive Control, scheduling, distributed optimization.**

I. Introduction

INFORMATION and Communication technologies have pervaded our everyday lives. Thanks to them, we enjoy a great variety of mobile apps while we are on the go, such as exchanging audio and video content and having our voice recognized in a snap. New applications, such as extended reality and autonomous driving are on their way and all of these apps generate a great amount of data. International Data Corporation (IDC) predicts that the yearly amount of data generated worldwide will grow 5 times, and that the percentage of real-time data generated by connected devices will reach 30% of the global data volume by 2025 (it was 15% in 2017) [1]. Most of such data has to be processed by either cloud or network servers. In an attempt to make this processing more efficient, a paradigm shift is occurring, going from a centralized *mobile cloud computing* (MCC) model towards a highly distributed *multi-access edge computing* (MEC) one, where computing power, network control, and storage are pushed to the network edge [2]. What it is often ignored, is that this computation drains a noticeable amount of energy, drastically increasing the carbon footprint of mobile networks [3] and MEC alone is unable to solve the energy consumption issue.

To ameliorate this, in the present work we advocate the use of off-grid renewable energy, such as solar radiation and wind, as a means to reduce the environmental impact of modern Information Technology (IT) systems [2], and we devise new online scheduling techniques for in-network computing tasks that are green by design. Specifically, we consider *energy-hybrid* MEC networks, where edge servers are installed at the base stations (at the network edge) and are co-powered by renewable energy sources and by the power grid. Any renewable energy surplus can be either stored, via local energy storage devices (batteries) or sold back to the grid. Computing jobs, subject to execution deadlines, are generated by access nodes (mobile users), and our chief objective is to execute them by minimizing the amount of energy that is purchased over time from the power grid, while meeting all deadlines. Computing servers can exchange workload (in full or in part) with their neighbors to relieve congested nodes. To allocate computing resources at runtime, an *online* approach based on *Model Predictive Control (MPC)* [4] with lookahead capabilities is devised, where external (exogenous) processes such as renewable energy and job arrivals are estimated within a prediction window, and their estimates are used to drive the online optimization of job schedules. A fully distributed solver for the job scheduling problem is devised, whereby network servers iteratively solve simpler *local sub-problems* communicating with their immediate neighbors, using the alternating direction method of multipliers (ADMM) [5].

The main contributions of the present work are:

i. energy efficiency is considered at all stages of our design, computing servers are equipped with batteries for energy storage, have access to renewable energy sources, and are connected to the power grid.
ii. Our system's model accounts for transmission and computing resources under arbitrary network topologies. It supports the processing of time-sensitive computing jobs with strict execution deadlines, and workload redistribution (load balancing).
iii. Two objective functions are devised, promoting contrasting resource allocation policy behaviors, namely *consolidation* (using as few servers as possible) and *load-balancing* (spreading the load across servers). To allocate resources at runtime, we propose an *online* approach

All authors are with the Department of Information Engineering at the University of Padova, via Gradenigo 6/b, 35131, Padova, Italy.

Michele Rossi is also with the Department of Mathematics "Tullio Levi-Civita" at the University of Padova, via Trieste 63, 35121, Padova, Italy.

*Corresponding author, email: giovanni.perin.2@phd.unipd.it.

Manuscript received February 18, 2021; revised ****** **, 20**.

based on *Model Predictive Control (MPC)*, which uses future job and energy arrival estimates, obtained via low-complexity predictors.

iv. For scalability purposes, a fully distributed iterative procedure for solving the predictive control problem is proposed, based on the Douglas-Rachford splitting (DRS) algorithm.

v. The effectiveness and benefits of our predictive control policy are evaluated thanks to an extensive simulation campaign, investigating the efficacy of the optimization goals, and the performance of the distributed algorithm in terms of convergence rate and overall amount of renewable energy utilized.

The proposed scheduling algorithm uses edge computing resources effectively, leading in the best case to a decrease of 50% in the amount of renewable energy that is sold to the power grid by heuristic policies, and that is instead utilized by the edge servers for processing. Other benefits include *server consolidation*, reducing by up to 40% the number of active servers across the edge network. These results are achieved by intelligently resorting to the cloud computing facility only when the edge processing capacity is exceeded, and by never violating execution deadlines.

The rest of the paper is organized as follows. Section II discusses the related works. Section III presents the system model. The optimization problem and its distributed solution are detailed in Sections IV and V, respectively. Numerical results are presented in Section VI, and concluding remarks are given in Section VII.

II. State of The Art

The multi-access edge computing paradigm has received considerable attention from academia [2], [3], [6]–[8] and industry [9] alike. The typical objectives pursued by MEC resource allocation algorithms are: minimizing the power consumption, minimizing the execution delay, or maximizing the revenue. In some studies, these objectives are combined, e.g., through a weighted sum of power consumption and delay [10]. In this work, we propose a distributed, online and adaptive optimization framework for computation load offloading in a network of edge servers, with the goal of managing effectively the energy coming from renewable sources. We consider CPU power, transmission bandwidth and execution deadline constraints in the formulation, as only a few works previously did, e.g., [11].

Server power consumption models and how they relate to CPU load are the focus of [12]–[14]. Reducing the power consumption of such systems is beneficial for lowering both the providers' costs and the environmental impact of the MEC infrastructure. Therefore, it is of paramount importance designing these systems to be as *energy-efficient* as possible [11], [15]. In [11], a model for the allocation of processing tasks in hierarchical MEC environments is proposed, by devising a distributed (ADMM-based) algorithm to solve the resource scheduling problem. Although the authors consider job execution deadlines, they propose a centralized strategy for inducing load-balancing, whereas, here, load-balancing and consolidation are a natural consequence of the chosen optimization function and are obtained via a fully distributed algorithm. Moreover, the setting of their work is static, while we propose an online and dynamic algorithm. That is, the optimization is continuously adapted to the time varying (exogenous) load and energy processes, exploiting a model predictive control (MPC) based framework. The authors of [15] propose a successive convex approximation (SCA) based algorithm to minimize the total mobile energy expenditure for offloading augmented reality tasks under latency constraints. They allocate communication and computation resources, considering the cloudlet as a single computation entity. In contrast, we consider an edge network of distributed agents. Furthermore, our goal does not corresponds to globally minimize the energy used by the system, but to optimally exploit the energy coming from renewable sources at each server. Note that the two objectives do not necessarily coincide as with renewables it is at times optimal to drain as much energy as possible, so as to minimize the energy that goes unused, and thus lost. According to [12]–[14], moreover, the CPU is energetically better exploited in high load conditions. In fact, under low load, most of the energy is drained to keep the CPU active. This is why we propose two optimization strategies, the first one for load-balancing (high load) and the other one for consolidation (low load) purposes.

Several works have considered the exploitation of renewable energy sources to power edge devices via energy harvesting (EH) technologies [3], [10], [16]–[19]. The authors of [10] devise an efficient reinforcement learning-based resource management algorithm. Their approach is online and adaptive and the goal is the correct management of the incoming energy arrivals. However, unlike what we do in the present work, their framework requires a centralized controller, which is a strict requirement in edge networks. The authors of [16] and [19] investigate a green MEC system with EH devices with equipped batteries, and develop an effective computation offloading strategy based on *Lyapunov optimization*. Their approach also belongs to the class of online and adaptive policies, but it is again centralized, and only the load-balancing objective is sought. In [3], an energy hybrid system is deployed, where mobile devices are equipped with EH capabilities, powered by the downlink channel. According to the heuristic scheme proposed by the authors, EH devices are fully capable of reliably powering a small-scale edge computing prototype system, during most (94.8%) of their experimental campaign. However, in their work, only end devices harvest energy from the environment, and this energy does not necessarily come from renewable sources. Conversely, in our work, edge servers are equipped with EH devices, and manage the incoming workload from end devices in their coverage area accordingly. Nonetheless, as also highlighted in [16] and [19], renewable energy sources are unreliable and often inadequate for fully supporting telecommunication networks demand. To cope with this, we propose a hybrid-energy model integrating them with the power grid, as done in [20].

MPC-based approaches have been previously used for controlling networked systems [20]–[24]. Data centers virtual machine (VM) management is the focus of [21], which presents an energy-aware consolidation strategy. A resource management approach effectively capturing the non-linear behavior in VM resource usage through fuzzy modelling is presented in [22]. The problem proposed in their work, however, is NP-hard, and authors use a genetic algorithm to retrieve a satisfactory solution. Besides being a centralized approach, it is also complex to solve. On the contrary, we use a fast version of ADMM, finding solutions even for non-convex problems and establishing a distributed implementation. The authors of [20] investigate how the monetary cost incurred in the energy purchases from the power grid can be minimized by dispatching the computation jobs to those servers that have enough energy and computation resources. However, they neither consider execution deadlines for jobs, nor they propose a strategy to solve the problem in a distributed fashion.

On a related note, the distributed energy resource scheduling problem of a set of smart homes is tackled in [23] by means of a stochastic MPC approach, with cooperation among neighbors.

III. System Model

We consider a MEC network comprising M computing entities or *edge servers*, $\mathcal{M} = \{1, \dots, M\}$, organized according to a given topology. Each node $i \in \mathcal{M}$ has a set of neighboring nodes, denoted by $\mathcal{N}_i$, to communicate with.

We identify with $\mathcal{M}_a \subset \mathcal{M}$ the set of *access nodes*, i.e., those servers co-located with base stations (BSs) that receive job processing requests from end users. An access node $i \in \mathcal{M}_a$ receiving a computing request can either process it *locally* or transfer such request (or a portion of it) to one or multiple neighboring nodes. Once the job's execution is completed, the computation result is sent back to the user terminal that originated the request. In this work, jobs are characterized by the computational effort they require, and by the time available for their execution. Edge servers can partially offload jobs multiple times during the temporal window available to execute such tasks, acting as relays. This amounts to a processing model where access nodes can:

1) process the workload locally, possibly splitting the task execution over the available time window;
2) partially outsource jobs to neighboring servers, that will have to complete the execution within the deadline.

In this way, the workload can be allocated across different servers, so that highly congested nodes can relieve themselves by partially outsourcing the execution of the load towards more capable or less congested ones.

An in-depth description of the job gathering process, as well as the execution and offloading duties is illustrated in the following sections. For simplicity, we assume that the system evolves according to a discrete-time model, with slots of length Δ and indexed by variable $k = 0, 1, \dots$. The notation used throughout the paper is summarized in Table I, whereas the main blocks of the server architecture are shown in Fig. 1.

A. Jobs gathered at access nodes

Computing tasks are collected by the access nodes $i \in \mathcal{M}_a$ and are locally executed or, alternatively, offloaded to other edge nodes. We abstract these generation processes by aggregating computing tasks received by access node i with the same execution deadline d at the beginning of time slot k into variable $t^d_{i,k}$. We also consider a maximum deadline of D slots, and, accordingly, we denote by $\boldsymbol{t}_{i,k} = [t^1_{i,k}, \dots, t^d_{i,k}, \dots, t^D_{i,k}]^T$ the column vector containing the cumulative generated workload for all possible execution deadlines $d \in \mathcal{D} = \{1, 2, \dots, D\}$ at node i. Note that the generated workload $\boldsymbol{t}_i$ enters the system at the beginning of a new time slot and, for the control framework, can either be considered as a disturbance or estimated.

TABLE I: List of symbols used in the paper.

Notation	Description
k	time slot index
N	MPC prediction horizon (time slots)
Δ	duration of a time slot
$\mathcal{D}$	set of execution deadlines ($\lvert\mathcal{D}\rvert = D$)
$\mathcal{M}$	set of edge servers ($\lvert\mathcal{M}\rvert = M$)
$\mathcal{M}_a$	set of access nodes
$\mathcal{N}_i$	set of neighbors of server $i \in \mathcal{M}$
$\mathcal{T}$	set of time slots ($k \in \mathcal{T}$, $\lvert\mathcal{T}\rvert = N$)
$s^d_{i,k}$	state of buffer d within edge server i at time k
$\bar{s}_i$	maximum amount of buffered workload in i
$e_{i,k}$	amount of energy stored at server i (energy buffer state)
$\bar{b}_i$	maximum battery capacity at server i
$\underline{b}_i$	battery capacity threshold at server i
$g_{i,k}$	amount of energy exchanged with the power grid by i
$o^d_{ij,k}$	CPU cycles with deadline d to be outsourced from i to j
$\overline{o}_{ij}$	maximum amount of workload exchanged from i to j
$w^d_{i,k}$	CPU cycles with deadline d allocated to CPU
$\bar{w}_i$	maximum computing power of server i (CPU cycles/slot)
G, O, W	dimensions of variables $\boldsymbol{g}$, $\boldsymbol{o}$, and $\boldsymbol{w}$, respectively
$h_{i,k}$	amount of energy harvested by node i during slot k
$t^d_{i,k}$	tasks (in CPU cycles) with deadline d generated in i
δ^{E}_i	discount factor accounting for energy decay
F	Jain's fairness index, defined in (37)
ϵ	Job generation rate
$\phi_{i,k}$	load factor at server i at time k, defined in (15)
η	efficiency metric, defined in (38)
$\mathcal{P}_\ell$	proximity operators, $\ell \in \{1, 2\}$
$\mathcal{Q}_\ell$	reflected proximity operators, $\ell \in \{1, 2\}$
T_m	edge network tier, $m \in \{0, 1, 2\}$
$\xi^{\mathrm{OFF}}_{i,j}$, ξ^{CPU}_i	workload to energy consumption conversion factors
$\xi^{\mathrm{PUR}}_{i,k}$, $\xi^{\mathrm{SOLD}}_{i,k}$	cost of purchasing (PUR) and selling (SOLD) energy

B. Edge server architecture

Each edge server has a central processing unit (CPU) whose computational power is shared by:

i. newly generated jobs which are neither offloaded nor backlogged (in case of an access node),
ii. the jobs offloaded from neighboring nodes, and
iii. backlogged jobs (previously arrived and temporarily kept in a queue for later processing).

Further, jobs can be partially offloaded or backlogged, and they also have to meet a certain execution deadline. This suggests grouping the load at edge servers according to the

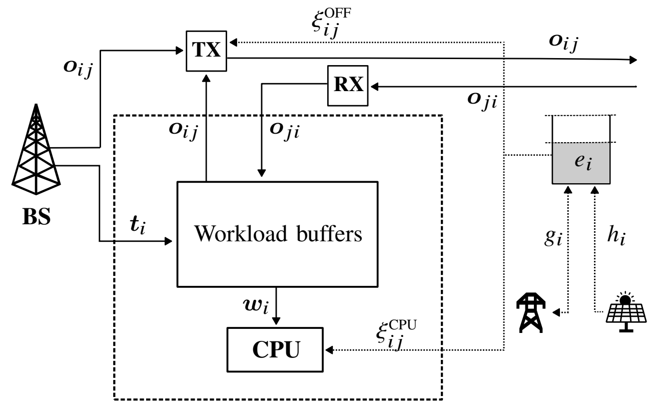


Fig. 1: **Edge server architecture**. Edge servers are co-located with the access points at BSs, and directly receive jobs from end devices. The servers can process the incoming workload locally or offload it to other servers. They are also equipped with a battery, which is charged by the energy harvested from the environment (h_i). Besides, the energy unit is also connected to the power grid, allowing the purchase and the sale of energy (g_i).

remaining slots available before the deadline d. That is, the buffered workload is organized according to its remaining lifetime d before the deadline expires. We conveniently model an edge server using D buffers, where each buffer is devoted to meeting a specific deadline d, as depicted in Fig. 2. Accordingly, we define as $s_{i,k}^d$ the buffer's *state*, corresponding to the backlogged jobs for edge server $i \in \mathcal{M}$ with deadline d at the beginning of time slot k, and we collect this information in column vectors $\boldsymbol{s}_{i,k} = [s_{i,k}^1, \dots, s_{i,k}^d, \dots, s_{i,k}^D]^T$.

The time evolution of vector $\boldsymbol{s}_{i,k}$ depends on whether the jobs (or portions thereof) are executed locally, if they are backlogged, or transferred elsewhere, as well as on the locally generated jobs $\boldsymbol{t}_{i,k}$. To precisely model these interactions we distinguish between:

a) **Local execution**: edge server i must decide the amount of workload to process at each time slot k: we denote by $w_{i,k}^d$ the amount of CPU cycles requested by tasks collected in buffer d that edge server i locally processes in time slot k.
b) **Offloading**: edge server i can offload a portion of a backlogged task to a neighboring node: we denote by $o_{ij,k}^d$ the amount of computing tasks (CPU cycles) that edge server i transfers to its neighbor j from buffer d during time slot k. We also assume that the expiring backlogged workload, with deadline $d = 1$, cannot be offloaded.

Hence, the equation governing the buffer state evolution at server i from time k to $k+1$ reads as

$$s_{i,k+1}^{d-1} = s_{i,k}^d + \underbrace{t_{i,k}^d}_{\text{locally generated}} - \overbrace{w_{i,k}^d}^{\text{locally executed}} + \underbrace{\sum_{j \in \mathcal{N}_i} o_{ji,k}^d}_{\text{offloaded here}} - \overbrace{\sum_{j \in \mathcal{N}_i} o_{ij,k}^d}^{\text{offloaded elsewhere}}, \tag{1}$$

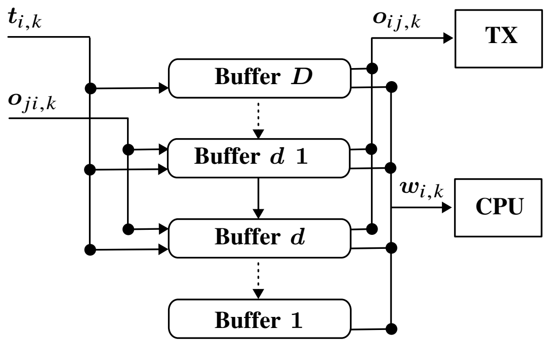


Fig. 2: **Detail of workload buffers**. At each server i, workload can be received from a neighbor ($\boldsymbol{o}_{ji,k}$) or generated locally ($\boldsymbol{t}_{i,k}$). The incoming workload is sent to the appropriate buffer, according to its execution deadline. Workload $\boldsymbol{w}_{i,k}$ is executed locally, whereas $\boldsymbol{o}_{ij,k}$ is offloaded to a neighbor server.

for $d \in \mathcal{D}/\{1\}$, and where we further assume that $s_{i,k+1}^D$ is only fed with locally generated workload. Note that in (1) we explicitly account for the tasks locally generated at the access node, $t_{i,k}^d$ where $t_{i,k}^d = 0$ if $i \notin \mathcal{M}_a$, the tasks that are locally executed, $w_{i,k}^d$, those incoming from neighboring nodes, $o_{ji,k}^d$, and the amount of workload that is outsourced $o_{ij,k}^d$.

By exploiting the *shift-by-one-position* binary matrix

$$\boldsymbol{T}_D = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & 0 & 1 \\ & & & 0 \end{bmatrix} \tag{2}$$

of size $D \times D$, having ones in the sup-diagonal and zeros elsewhere, (1) can be compactly rewritten in vector form as

$$\boldsymbol{s}_{i,k+1} = \boldsymbol{T}_D \left(\boldsymbol{s}_{i,k} + \boldsymbol{t}_{i,k} - \boldsymbol{w}_{i,k} + \sum_{j \in \mathcal{N}_i} \left(\boldsymbol{o}_{ji,k} - \boldsymbol{o}_{ij,k} \right) \right), \tag{3}$$

where $\boldsymbol{w}_{i,k}$, $\boldsymbol{o}_{ij,k}$, and $\boldsymbol{o}_{ji,k}$ are vectors collecting all the offloading and local processing decision variables for the D buffers. Refer to Fig. 2 for a pictorial representation of the relations between these variables.

C. Energy consumption model

Edge nodes are equipped with *energy harvesting* capabilities. In particular, each node can collect energy from a renewable energy source and store it onto a local energy buffer. Due to the unreliable nature of such energy sources, each edge node is also connected to a power grid, from which it can purchase the energy needed to support its operations at all times, or even sell excess energy.

The energy buffer state $e_{i,k}$, $k = 0, 1, \dots$, at node $i \in \mathcal{M}$ evolves according to

$$e_{i,k+1} = \delta_i^{\mathrm{E}} e_{i,k} + h_{i,k} + g_{i,k} - \sum_{j \in \mathcal{N}_i, d \in \mathcal{D}} \xi_{ij}^{\mathrm{OFF}} o_{ij,k}^d - \sum_{d \in \mathcal{D}} \xi_i^{\mathrm{CPU}} w_{i,k}^d \tag{4}$$

where:

- $\delta_i^{\mathrm{E}} \in [0,1]$ is a discount factor accounting for the natural energy decay in the battery;
- $h_{i,k}$ represents the (random) amount of energy harvested from the environment, which is either known or, more realistically, estimated;
- $g_{i,k}$ represents the energy exchanged with the grid, namely, purchased if $g_{i,k} > 0$, or sold to the grid if $g_{i,k} < 0$;
- ξ_{ij}^{OFF} and ξ_i^{CPU} respectively represent the conversion factors between the offloaded and locally processed workloads, and the energy required by such processing.

D. Job and energy arrivals: dynamics and prediction

As for the characterization of the amount of harvested energy $h_{i,k}$ and aggregated amount of computing jobs $t_{i,k}^d$ that enter the system, we rely on (correlated) Markov Chains (MC) with parameters that can be customized per node i (energy/workload), and buffer d (workload only). Arrivals are modeled via two-state discrete time MC (`ON-OFF` behavior). Accordingly, each workload buffer d can receive at most one aggregate computing job per time slot. If at the beginning of time slot k the chain associated with buffer d at node i is in the `OFF` state, the job intensity is $t_{i,k}^d = 0$ for this buffer. Instead, in the `ON` state the job intensity $t_{i,k}^d$ is randomly picked from a state-specific discrete probability mass distribution (pmd). The harvested energy process is generated analogously.

Prediction is a key ingredient for an MPC framework. Next, we introduce three forecasting strategies, entailing different degrees of knowledge about the generation processes.

i. **Genie predictor** (ideal). Arrivals times and job intensities are known for all past and future time slots; this predictor is used to derive an upper bound on the achievable performance.
ii. **MC-unroll predictor**. This predictor knows the statistical model governing the job arrivals, i.e., the MC transition matrix, and also the pmd governing the intensities $t_{i,k}^d$. Then, a sequence of job arrival estimates for the future time slots $k+1, k+2, \dots$ is obtained as a realization of the corresponding MC over these future time slots, starting from the MC's current state. Arrival intensities $t_{i,k}^d$ are sampled from the pmd in the `ON` state.
iii. **i.i.d. predictor**. Let us define the average probability of observing an arrival (in any arbitrary time slot) as $\hat{f}_i^{\mathrm{MC}}$, and the average intensity of arrivals as $\overline{p}_i^{\mathrm{MC}}$ (expected pmd value). Once these quantities are known, or more realistically estimated, predictions over future time slots can be obtained as a realization of an i.i.d. process generating an arrival with probability $\hat{f}_i^{\mathrm{MC}}$ and with intensity $\overline{p}_i^{\mathrm{MC}}$, and no arrivals with probability $1 - \hat{f}_i^{\mathrm{MC}}$.

IV. Problem Formulation

A. Model predictive control framework

The system state equations (3) and (4) constitute the backbone for an MPC framework [4] which predicts the system state in the next N time steps (the *prediction horizon*), controlling the scheduling of the incoming and outgoing workloads, as well as the amount of energy purchased and sold. The computed input is applied at the immediate following step $k+1$ and the procedure is repeated again, updating the system state with new measures every time. This approach is known as *receding horizon*.

For notation simplicity, in the following we assume that the reference time slot is $k = 1$, so that the time slots of interest for MPC are those with $k \in \mathcal{T} = \{1, \dots, N\}$, and the implemented decisions are those at slot $k = 1$, namely, for each server i: a) the amount of workload to be executed locally $\boldsymbol{w}_{i,1}$; b) the workload $\boldsymbol{o}_{ij,1}$ to offload towards neighbor nodes $j \in \mathcal{N}_i$; and c) the amount of energy $g_{i,1}$ that is to be either purchased from or sold to the power grid.

Details on the optimization problem that must be solved under the said MPC framework are given in the following sections.

B. Workload buffers evolution

We preliminarily generalise (3) in a form that projects the buffers' state forward in time by n time slots. By iterated application of (3), we obtain

$$\boldsymbol{s}_{i,k+n} = \boldsymbol{T}_D^n \boldsymbol{s}_{i,k} + \sum_{m=0}^{n-1} \boldsymbol{T}_D^{n-m} \Bigg(\boldsymbol{t}_{i,k+m} - \boldsymbol{w}_{i,k+m} + \sum_{j \in \mathcal{N}_i} (\boldsymbol{o}_{ji,k+m} - \boldsymbol{o}_{ij,k+m}) \Bigg), \quad (5)$$

where control actions are taken and job arrivals occur during time slots $k, k+1, \dots, k+n-1$.

We compactly write (5) by stacking vectors and matrices over the prediction horizon N. To this aim, we collect buffers' states in the column vector $\boldsymbol{s} = \{\{\boldsymbol{s}_{i,k+1}\}_{k \in \mathcal{T}}\}_{i \in \mathcal{M}}$ in such a way that the information of node $i = 1$ is placed atop, followed by the information of node $i = 2$, etc. We do similarly for the newly generated workload, $\boldsymbol{t} = \{\{\boldsymbol{t}_{i,k}\}_{k \in \mathcal{T}}\}_{i \in \mathcal{M}}$, the locally executed workload, $\boldsymbol{w} = \{\{\boldsymbol{w}_{i,k}\}_{k \in \mathcal{T}}\}_{i \in \mathcal{M}}$, and the offloaded workload, $\boldsymbol{o} = \{\{\{\boldsymbol{o}_{ij,k}\}_{k \in \mathcal{T}}\}_{j \in \mathcal{N}_i}\}_{i \in \mathcal{M}}$. With this notation in mind, the buffer state evolution (5) over the prediction horizon, and for the whole network, is expressed by the linear relation

$$\boldsymbol{s} = \boldsymbol{A}_w \boldsymbol{s}_1 + \boldsymbol{B}_w (\boldsymbol{t} - \boldsymbol{w} + \boldsymbol{C}_w' \boldsymbol{o} - \boldsymbol{C}_w'' \boldsymbol{o}) , \quad (6)$$

where $\boldsymbol{s}_1 = \{\boldsymbol{s}_{i,1}\}_{i \in \mathcal{M}}$ collects the initial states of all buffers (*memory*), while $\boldsymbol{A}_w$, $\boldsymbol{B}_w$, $\boldsymbol{C}_w'$, and $\boldsymbol{C}_w''$ are appropriate matrices that can be deduced from (5). Specifically, $\boldsymbol{C}_w'$ and $\boldsymbol{C}_w''$ are binary matrices that respectively collect the sums for $j \in \mathcal{N}_i$ in (5). Instead, for matrices $\boldsymbol{A}_w$ and $\boldsymbol{B}_w$, we have

$$\boldsymbol{A}_w = \boldsymbol{I}_M \otimes \left(\sum_{k=1}^{N} \boldsymbol{e}_{k,N} \otimes \boldsymbol{T}_D^k \right), \quad (7)$$

$$\boldsymbol{B}_w = \boldsymbol{I}_M \otimes \left(\sum_{k=1}^{N} \boldsymbol{T}_N^{k-1} \otimes \boldsymbol{T}_D^k \right) \quad (8)$$

where $\otimes$ is the Kronecker matrix product, $\boldsymbol{I}_M$ is the identity matrix of order M, and $\boldsymbol{e}_{k,M}$ is its kth column.

We observe that, according to the MPC approach, in (6) the state variable $\boldsymbol{s}_1$ is assumed to be known, and, similarly, the newly generated workload $\boldsymbol{t}$ is assumed to be known in the reference time slot (entries $\boldsymbol{t}_{i,1}$) and estimated for future time slots (entries $\boldsymbol{t}_{i,k}$ for $k > 1$). When estimates for future generated workload are not available, they can be neglected and treated as disturbances, i.e., $\boldsymbol{t}_{i,k} = 0$ for $k > 1$. Instead, the locally executed workload, $\boldsymbol{w}$, and the offloaded workloads, $\boldsymbol{o}$, play the role of optimization variables.

A number of side constraints that govern the behavior of (6) further need to be formalized.

Positive bounds: workloads are, by definition, positive quantities, that is

$$w^d_{i,k}, o^d_{ij,k} \geq 0 \tag{9}$$

for each node $i \in \mathcal{M}$, neighbor node $j \in \mathcal{N}_i$, and buffer $d \in \mathcal{D}$, and through the entire time span $k \in \mathcal{T}$.

Offloading bounds: the amount of workload exchanged between nodes can be upper bounded (e.g., due to the physical transmission constraints). We therefore assume

$$\sum_{d \in \mathcal{D}/\{1\}} o^d_{ij,k} \leq \bar{o}_{ij} \tag{10}$$

for each node $i \in \mathcal{M}$, neighbor node $j \in \mathcal{N}_i$, and slot k.

Buffer bounds: the amount of workload buffered at each node $i \in \mathcal{M}$ is bounded as

$$0 \leq s^d_{i,k} \leq \bar{s}_i \,. \tag{11}$$

Workload conservation: for each node $i \in \mathcal{M}$ and buffer $d \in \mathcal{D}/\{1\}$ the workload conservation inequality

$$w^d_{i,k} + \sum_{j \in \mathcal{N}_i} o^d_{ij,k} \leq s^d_{i,k} + t^d_{i,k} \tag{12}$$

applies, which ensures that nodes can process or offload only the existing workload, i.e, the backlogged ($s^d_{i,k}$) or the newly generated ($t^d_{i,k}$) one. This corresponds to the assumption that the offloaded workload $o^d_{ij,k}$ takes a full time slot to be delivered to node j, and it will therefore be available for execution (in buffer $d-1$) starting from time $k+1$. Thus, workload offloading delays execution of at least one time slot.

Forced execution: as stated above, we force edge nodes to process expiring tasks, i.e., tasks with just a time slot available before their deadline expires. Hence, workload offloading from buffer $d = 1$ is prohibited. We therefore assume

$$w^1_{i,k} = s^1_{i,k} + t^1_{i,k}\,, \quad o^1_{ij,k} = 0 \tag{13}$$

for every node $i \in \mathcal{M}$, neighbor node $j \in \mathcal{N}_i$, and slot k.

Processing capacity: if we assume that an edge server has finite computing capabilities, that is, a total computational power of $\bar{w}_i$ CPU cycles per time slot, then it must hold

$$\sum_{d \in \mathcal{D}} w^d_{i,k} \leq \bar{w}_i \tag{14}$$

for every edge server $i \in \mathcal{M}$ and time slot k. Based on the processing capacity, the *load factor* of server i in slot k is

$$\phi_{i,k} = \frac{\sum_{d \in \mathcal{D}} w^d_{i,k}}{\bar{w}_i} \,. \tag{15}$$

Note that $0 \leq \phi_{i,k} \leq 1$.

C. Energy buffers evolution

The evolution of the energy buffers (4) can be tracked similarly. In this case, the vectors of interest are the energy buffer states $\boldsymbol{e} = \{\{e_{i,k+1}\}_{k\in\mathcal{T}}\}_{i\in\mathcal{M}}$, the initial state $\boldsymbol{e}_1 = \{e_{i,1}\}_{i\in\mathcal{M}}$, the harvested energy vector $\boldsymbol{h} = \{\{\boldsymbol{h}_{i,k}\}_{k\in\mathcal{T}}\}_{i\in\mathcal{M}}$, and the exchange with the grid $\boldsymbol{g} = \{\{\boldsymbol{g}_{i,k}\}_{k\in\mathcal{T}}\}_{i\in\mathcal{M}}$. The evolution over the prediction horizon is again expressed as a linear relation,

$$\boldsymbol{e} = \boldsymbol{A}_e \boldsymbol{e}_1 + \boldsymbol{B}_e(\boldsymbol{h} + \boldsymbol{g} - \boldsymbol{C}'_e \boldsymbol{o} - \boldsymbol{C}''_e \boldsymbol{w}) \tag{16}$$

where $\boldsymbol{g}$, $\boldsymbol{o}$, and $\boldsymbol{w}$ play the role of optimization variables, $\boldsymbol{e}_1$ is known, and $\boldsymbol{h}$ is known at time slot $k = 1$ and is estimated for $k > 1$. In addition, the matrices $\boldsymbol{A}_e$ and $\boldsymbol{B}_e$ in (16) assume a form similar to that of (7), namely

$$\boldsymbol{A}_e = \boldsymbol{I}_M \otimes \left(\sum_{k=1}^{N} (\delta^{\mathrm{E}}_i)^k \, \boldsymbol{e}_{k,N} \right) \tag{17}$$

$$\boldsymbol{B}_e = \boldsymbol{I}_M \otimes \left(\sum_{k=1}^{N} (\delta^{\mathrm{E}}_i)^k \, \boldsymbol{T}^{k-1}_N \right) \tag{18}$$

while $\boldsymbol{C}'_e$ and $\boldsymbol{C}''_e$ are binary matrices collecting the two sums of (4). The only constraint that is needed in this case is the following one.

Energy bounds: we require that the energy buffer updates $e_{i,k+1}$ be limited by the maximum battery capacity $\bar{b}_i$, and a minimum working threshold $\underline{b}_i$, that is

$$\underline{b}_i \leq e_{i,k+1} \leq \bar{b}_i \tag{19}$$

for every edge server $i \in \mathcal{M}$ and time slot $k \in \mathcal{T}$.

D. Objective functions

As aforementioned, MPC solves at every time step an optimization problem for every node $i \in \mathcal{M}$, minimizing some predefined cost which is function of the current measured state and of the inputs to be optimized. We identify this cost with $J(\boldsymbol{g}, \boldsymbol{o}, \boldsymbol{w})$ to underline its dependence on the global optimization variables. For the same reason, (6) will be denoted more explicitly in the form $\boldsymbol{s}(\boldsymbol{o}, \boldsymbol{w})$, and (16) in the form $\boldsymbol{e}(\boldsymbol{g}, \boldsymbol{o}, \boldsymbol{w})$. This leads to the *centralized* optimization problem

$$\boxed{\begin{aligned} P_1 : \quad & \min_{\boldsymbol{g},\boldsymbol{o},\boldsymbol{w}} J(\boldsymbol{g}, \boldsymbol{o}, \boldsymbol{w}) \qquad (20) \\ & \text{s.t. (9)} - \text{(14), and (19) hold,} \end{aligned}}$$

to be solved to find the resource allocation over the prediction horizon N.

In this work, two processing cost functions are compared: a quadratic and a logarithmic one, the latter leading to a non-convex optimization problem. The importance of the workload buffers cost minimization with respect to the operation cost is tuned through a parameter $\gamma \in [0, 1]$.

Quadratic cost function: Writing the dependency on the time slot k as a subscript, the quadratic cost is compactly expressed as

$$J(\boldsymbol{g}, \boldsymbol{o}, \boldsymbol{w}) = (1-\gamma)\|\boldsymbol{s}(\boldsymbol{o}, \boldsymbol{w})\|^2_{\boldsymbol{Q}_s} + \gamma\Big(\mathcal{R}_e(\boldsymbol{g}) + \|\boldsymbol{o}\|^2_{\boldsymbol{R}_o} + \langle \boldsymbol{r}_o, \boldsymbol{o} \rangle + \|\boldsymbol{w}\|^2_{\boldsymbol{R}_w}\Big), \tag{21}$$

where $\|\boldsymbol{x}\|_{\boldsymbol{Q}}^2 = \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$ denotes a weighted norm, and $\langle\cdot,\cdot\rangle$ denotes the inner product. All matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ are positive semidefinite (and, typically, diagonal), and collect the workload buffering cost ($\boldsymbol{Q}$), and the energy cost of transmissions ($\boldsymbol{R}_o$) and processing ($\boldsymbol{R}_w$), respectively. The linear term with cost $\boldsymbol{r}_o$ is an $L1$-norm penalty on transmissions that is added to promote a spare use of channel resources. The cost $\mathcal{R}_e(\boldsymbol{g})$ of the energy exchanged with the power grid takes the piecewise linear form

$$\mathcal{R}_e(\boldsymbol{g}) = \sum_{i\in\mathcal{M},k\in\mathcal{T}} \mathcal{R}_{i,k}(g_{i,k}) \ , \quad \mathcal{R}_{i,k}(x) = \begin{cases} \xi_{i,k}^{\text{PUR}}\,x, & x \geq 0 \\ -\xi_{i,k}^{\text{SOLD}}\,x, & x < 0 \end{cases} \tag{22}$$

where we assume $\xi_{i,k}^{\text{SOLD}} < \xi_{i,k}^{\text{PUR}}$, to prioritize the use of the energy harvested over that purchased from the power grid.

The choice of a quadratic operation cost naturally induces load balancing in the system, acting as $L2$-norm regularizer. Note that, under the cost function (21), P_1 can be solved in a centralized manner using a constrained quadratic programming (QP) solver. As a matter of fact, the constraints in (20) are all linear, all the weighted norms in (21) have a quadratic expression thanks to the fact that (6) and (16) are linear, and the energy cost in (22) can be expressed in a linear form by separating $\boldsymbol{g}$ into its positive and negative contributions, that is $\boldsymbol{g} = \boldsymbol{g}^+ - \boldsymbol{g}^-$ and $\mathcal{R}_e(\boldsymbol{g}) = \mathcal{R}_e(\boldsymbol{g}^+) + \mathcal{R}_e(-\boldsymbol{g}^-)$, with the additional linear constraints $\boldsymbol{g}^+ \geq \boldsymbol{0}$ and $\boldsymbol{g}^- \geq \boldsymbol{0}$.

Logarithmic cost function: The intuition behind the choice of a logarithmic (non-convex) cost function is that it directly produces *sparse* solutions, promoting server consolidation. Because this function is superlinear in proximity of the zero, a sleeping server will prefer to offload the workload to an already working neighbor if it can avoid to turn its processing unit on. With the same notation employed for the quadratic cost, the logarithmic objective function is defined as

$$J(\boldsymbol{g},\boldsymbol{o},\boldsymbol{w}) = (1-\gamma)\langle \boldsymbol{q}_s, \boldsymbol{s}(\boldsymbol{o},\boldsymbol{w})\rangle + \gamma\Big(\mathcal{R}_e(\boldsymbol{g}) + \langle \boldsymbol{r}_o,\boldsymbol{o}\rangle + \sum_{i\in\mathcal{N}} \log(1+\langle \boldsymbol{r}_{w,i},\boldsymbol{w}_i\rangle)\Big), \tag{23}$$

where $\boldsymbol{q}_s$ is the vector of workload buffers costs, whereas $\boldsymbol{r}_o$ and $\boldsymbol{r}_w$ respectively represent the energy cost of transmitting and processing a unit of workload. Note that the cost function in (23) is non-convex and, in turn, heuristic methods must be used to solve the associated optimization problem.

V. Distributed Solution

A. Background on Distributed Optimization

In the present work we use *message passing* techniques, as they nicely suit the considered distributed network setup. In these frameworks, network agents exchange partial computation outputs with their immediate (one hop) neighbors using (sub)gradient methods, where local gradient descent steps are combined with consensus averaging. Among these techniques, ADMM [5] has recently received a considerable attention as an effective method to solve networked optimization problems by iteratively applying simple optimization steps, while ensuring good convergence properties (speed and stability) at both local and global level [25]–[27]. ADMM is in close relation with, and in many cases equivalent to, a large number of alternative approaches, e.g., *Douglas-Rachford splitting*, *proximal point* algorithms, and the *split Bregman* algorithm [25], [26]. With these algorithms, the required level of coordination among network agents depends on factors such as the considered decomposition strategy and the underlying graph (communication) topology [27], [28]. Additionally, ADMM-like strategies can be heuristically used to deal with non-convex problems [5], [29].

Distributed MPC. Controlling networked systems of agents (servers) is common to many engineering problems of interest, and previous work investigating distributed procedures for solving MPC (here referred to as *distributed MPC* (D-MPC)) can be found in [28], [30], [31]. In [30], different algorithms are compared in terms of convergence speed, number of messages exchanged and distributed computation architecture. The authors of [28] propose an ADMM-based algorithm to solve the D-MPC problem, and the effects of prematurely terminating the iterative ADMM procedure are investigated in [31].

In this work, we derive a customized DRS of the centralized MPC problem defined in (20), optimizing a convex and a non-convex cost function in a distributed fashion over a given topology.

B. ADMM framework for scheduling computing jobs

The centralized problem P_1 in (20) can be solved in a distributed fashion, provided that we split the cost function and the constraints into node-dependent contributions that rely on separate entries (i.e., a partition) of the global optimization variables. This can be obtained by duplicating the offloaded workloads $\boldsymbol{o}$ in the pair $(\boldsymbol{o},\tilde{\boldsymbol{o}})$, where $\tilde{\boldsymbol{o}} = \boldsymbol{o}$, that is, by considering the optimization vector

$$\boldsymbol{x} = [\boldsymbol{g};\boldsymbol{o};\tilde{\boldsymbol{o}};\boldsymbol{w}] \tag{24}$$

and the associated sub-space

$$\mathcal{X} = \{\boldsymbol{x}|\tilde{\boldsymbol{o}} = \boldsymbol{o}\} \ , \tag{25}$$

which identifies an added linear constraint to be solved. Accordingly, the cost functions and the linear constraints in (20) must be slightly modified in order to obtain the said separation. This simply requires replacing $o_{ij,k}^d$ with $\tilde{o}_{ij,k}^d$ throughout the expressions of Section IV, in such a way that what is offloaded *elsewhere* is associated with variables $\boldsymbol{o}$, and what is offloaded *from somewhere else* is associated with the duplicated variables $\tilde{\boldsymbol{o}}$. With this idea in mind, the state update expression (6) assumes the alternative form

$$\boldsymbol{s}(\boldsymbol{x}) = \boldsymbol{A}_w\boldsymbol{s}_1 + \boldsymbol{B}_w(\boldsymbol{t} - \boldsymbol{w} + \boldsymbol{C}'_w\tilde{\boldsymbol{o}} - \boldsymbol{C}''_w\boldsymbol{o}) \ , \tag{26}$$

while the energy buffer update (16) is not modified (since it does not use $\tilde{\boldsymbol{o}}$). Also, the constraints (9)-(14) and (19), as well as the cost expressions (21) and (23), are not modified but for the fact that state variables now exploit (26). Hence, problem P_1 can be rewritten in the equivalent form

$$
\boxed{\begin{aligned} P_2: \quad & \min_{\boldsymbol{x}} J(\boldsymbol{x}) && (27)\\ & \text{s.t. } \boldsymbol{x} \in \mathcal{X} \\ & \quad \boldsymbol{x} \in \mathcal{Y} = \{\boldsymbol{x} | (9) - (14) \text{ and } (19) \text{ hold}\} \end{aligned}}
$$

where the linear constraints of problem (20) are collected in the (linear) sub-space $\mathcal{Y}$.

C. ADMM algorithm

P_2 in (27) is a large scale optimization problem whose complicating constraints $\boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{x} \in \mathcal{Y}$ make it non-separable in simple local sub-problems. However, it can be parallelised by duplicating variable $\boldsymbol{x}$ in a form that separates the complicating constraints, and that is amenable to distributed implementation by using ADMM.

Specifically, the reference problem for distributed processing is equivalently rewritten in the form

$$
\boxed{\begin{aligned} P_3: \quad & \min_{\boldsymbol{x}} \underbrace{J(\boldsymbol{x}_1) + \mathcal{I}_{\mathcal{Y}}(\boldsymbol{x}_1)}_{f_1(\boldsymbol{x}_1)} + \underbrace{\mathcal{I}_{\mathcal{X}}(\boldsymbol{x}_2)}_{f_2(\boldsymbol{x}_2)} && (28)\\ & \text{s.t. } \boldsymbol{x}_1 = \boldsymbol{x}_2 \end{aligned}}
$$

where

$$
\mathcal{I}_{\mathcal{X}}(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} \in \mathcal{X} \,, \\ +\infty & \text{otherwise} \,. \end{cases} \tag{29}
$$

is the indicating function of set $\mathcal{X}$.

Solution to (28) is here obtained via ADMM, which finds a saddle point of the associated augmented Lagrangian through a minimization that alternates between the direction of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. The specific approach that we use is the so-called scaled variable ADMM (e.g., see [25], [26]), illustrated in Algorithm 1, where the $^+$ sign denotes an updated variable, ρ and q are tuneable parameters that control the convergence speed, and

$$
\mathcal{P}_\ell(\boldsymbol{u}) = \operatorname*{argmin}_{\boldsymbol{x}} \; f_\ell(\boldsymbol{x}) + \frac{\rho}{2}||\boldsymbol{x} - \boldsymbol{u}||^2 \,, \tag{30}
$$

with $\ell = 1, 2$, are *proximity* operators.

Algorithm 1 Scaled variables ADMM

```
1: $x_1^+ = \mathcal{P}_1(x_2 - \lambda)$              minimize with respect to $x_1$
2: $y^+ = 2q x_1^+ + (1-2q) x_2$                            scale variables
3: $x_2^+ = \mathcal{P}_2(y^+ + \lambda)$              minimize with respect to $x_2$
4: $\lambda^+ = \lambda + y^+ - x_2^+$                update Lagrange multipliers
```

Nicely, the proximity operator (30) with $\ell = 2$ assumes the simple form

$$
\mathcal{P}_2(\boldsymbol{u}) = \operatorname*{argmin}_{\boldsymbol{x}\in\mathcal{X}} \; \frac{\rho}{2}||\boldsymbol{x} - \boldsymbol{u}||^2 = \boldsymbol{L}_{\mathcal{X}} \cdot \boldsymbol{u} \tag{31}
$$

where $\boldsymbol{L}_{\mathcal{X}}$ is the *projector* associated with the subspace $\mathcal{X}$, which extracts the component of $\boldsymbol{u}$ that lies on $\mathcal{X}$, and, accordingly, removes the component orthogonal to $\mathcal{X}$. We have

$$
\boldsymbol{L}_{\mathcal{X}} = \begin{bmatrix} \boldsymbol{I}_G & & & \\ & \frac{1}{2}\boldsymbol{I}_O & \frac{1}{2}\boldsymbol{I}_O & \\ & \frac{1}{2}\boldsymbol{I}_O & \frac{1}{2}\boldsymbol{I}_O & \\ & & & \boldsymbol{I}_W \end{bmatrix} , \tag{32}
$$

where G, O, and W are the dimensions (lengths) of, respectively, $\boldsymbol{g}$, $\boldsymbol{o}$, and $\boldsymbol{w}$. Note that (32) is a projection operator that allows extracting an average value from $\boldsymbol{o}$ and $\tilde{\boldsymbol{o}}$.

Instead, the proximity operator (30) with $\ell = 1$ takes the form

$$
\mathcal{P}_1(\boldsymbol{u}) = \operatorname*{argmin}_{\boldsymbol{x}\in\mathcal{Y}} \; J(\boldsymbol{x}) + \frac{\rho}{2}||\boldsymbol{x} - \boldsymbol{u}||^2 \tag{33}
$$

which, thanks to choice (24), is a separable form that corresponds to a number of local problems of reduced dimension, which can be solved in parallel. Specifically, the local problem at the ith edge server uses the $(1 + (1 + 2|\mathcal{N}_i|)D)N \simeq 2|\mathcal{N}_i|DN$ variables $\boldsymbol{x}_i = \{g_{i,k}, \{\boldsymbol{o}_{ij,k}, \tilde{\boldsymbol{o}}_{ji,k}\}_{j\in\mathcal{N}_i}, \boldsymbol{w}_{i,k}\}_{k\in\mathcal{T}}$, which store the information available locally, and can be written in the form

$$
\mathcal{P}_{1,i}(\boldsymbol{u}) = \arg\min_{\boldsymbol{x}_i\in\mathcal{Y}_i} \; J_i(\boldsymbol{x}_i) + \frac{\rho}{2}||\boldsymbol{x}_i - \boldsymbol{u}_i||^2 \,, \tag{34}
$$

where $\mathcal{Y}_i$ collects the constraints (9)-(14), and (19) for the ith node, and $J_i(\cdot)$ is the cost contribution ((21) or (23)) of node i.

D. Douglas-Rachford splitting version

By further defining the *reflected proximity* $\boldsymbol{Q}_\ell$ operators

$$
\boldsymbol{Q}_\ell(\boldsymbol{u}) = 2\sqrt{\rho}\mathcal{P}_\ell(\boldsymbol{u}/\sqrt{\rho}) - \boldsymbol{u} \,, \tag{35}
$$

Algorithm 1 can be equivalently rewritten in the extremely compact form of Algorithm 2, namely a DRS counterpart that tracks the unique variable $\boldsymbol{z} = \sqrt{\epsilon}(\boldsymbol{y}^+ + \boldsymbol{\lambda})$ (see details of this formalization in [26]). The explicit relation with the system variables is in this case

$$
\boldsymbol{x}_1^+ = \mathcal{P}_1(\boldsymbol{Q}_2(\boldsymbol{z})/\sqrt{\rho}) \,, \quad \boldsymbol{x}_2 = \mathcal{P}_2(\boldsymbol{z}/\sqrt{\rho}) \,. \tag{36}
$$

Algorithm 2 Douglas-Rachford Splitting counterpart

```
1: $z^+ = (1-q)z + q Q_1(Q_2(z))$
```

In Algorithm 2, $q \in [0, 1]$, $\boldsymbol{z}$ is the global version of the variables $\boldsymbol{z}_i = \{\gamma_{i,k}, \{\boldsymbol{\sigma}_{ij,k}, \tilde{\boldsymbol{\sigma}}_{ji,k}\}_{j\in\mathcal{N}_i}, \boldsymbol{\omega}_{i,k}\}$, where $\gamma_{i,k}$, $\boldsymbol{\sigma}_{ij,k}$, $\tilde{\boldsymbol{\sigma}}_{ji,k}$ and $\boldsymbol{\omega}_{i,k}$ respectively correspond to the projections of $g_{i,k}$, $\boldsymbol{o}_{ij,k}$, $\tilde{\boldsymbol{o}}_{ji,k}$ and $\boldsymbol{w}_{i,k}$, upon applying the transformation (36). Algorithm 2 works on a unique state variable $\boldsymbol{z}$, as opposed to the three state variables of Algorithm 1, namely, $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, and $\boldsymbol{\lambda}$, hence it is to be preferred from a computational perspective, also because the computational complexities of operators $\mathcal{P}_\ell$ and $\boldsymbol{Q}_\ell$ is identical. In addition, we empirically verified that, despite its simplicity (or, possibly, because of it), Algorithm 2 shows an improved numerical stability, which represents a further added value. Type-II Anderson acceleration is also added, to reduce the number of iterations for reaching convergence. It is a higher order acceleration technique that computes the new optimal direction of the variable $\boldsymbol{z}$ considering a linear combination of all the stored values back in time up to a certain memory limit (see A2DR [32] for further details).

Stopping criterion. The variable $\boldsymbol{z}$, which, as aforementioned, is the only one tracked, carries information from both the primal and the dual residuals. Therefore, a suitable method is to fix a desired threshold z_{obj}, and stop when $||\boldsymbol{z}^+ - \boldsymbol{z}|| < z_{\text{obj}}$. Besides evaluating the residuals, we also compute the current objective function value, and put a further threshold on the relative difference concerning the previous iteration: $\left|\frac{f(\boldsymbol{x}_1)}{f(\boldsymbol{x}_1^+)}\right| - 1 < f_{\text{obj}}$.

E. Convergence properties

Quadratic cost function: when the quadratic objective function of (21) is used, (33) amounts to solving a convex quadratic problem with linear constraints. Any QP solver can be used to obtain local updates and, moreover, the algorithm is ensured to converge linearly (e.g., see [25]) thanks to the fact that functions $f_\ell(\boldsymbol{x})$ are proper, lower semicontinuous, and convex. Interestingly, the convergence speed can be tracked by observing the behavior of $\|\boldsymbol{z}^+ - \boldsymbol{z}\|$, which is guaranteed to strictly decrease [26].

Logarithmic cost function: When, instead, the non-convex logarithmic cost function (23) is used, then the target function in (33) assumes a form proportional to $J(\boldsymbol{x}) = \boldsymbol{p}^T\boldsymbol{x} + \sum_{i\in\mathcal{M}} \log(1 + \boldsymbol{q}_i^T\boldsymbol{x})$, which is a concave function, for some values $\boldsymbol{p}$ and $\boldsymbol{q}_i$, $i \in \mathcal{M}$. In this case, a suitable method to control convergence is provided by [29], that uses DC programming with linear approximation. Specifically, convergence is ensured under weak assumptions if the cost function in (33) is replaced by its convex (actually, linear) part, using the first order Taylor expansion, namely, by $\tilde{J}(\boldsymbol{x}) = \boldsymbol{p}^T\boldsymbol{x} + \sum_{i\in\mathcal{M}} \boldsymbol{q}_i^T\boldsymbol{x}/(1 + \boldsymbol{q}_i^T\boldsymbol{x}_1/\sqrt{\rho})$, where we used the newly introduced (and compact) notation, and where we dropped the constant terms since they do not play any role in the minimization. With this approach in mind, the local problems are quadratic at every iteration and a QP solver can be used also in this case.

F. Final distributed algorithm

Algorithm 2 is expressed in compact form thanks to the reflected proximity operators defined in (35). The final algorithm is written from a server perspective in the below Algorithm 3. Lines 7-8 of Algorithm 3 represent the only local exchange of information that is required at each iteration, while the remaining operations are performed locally. Specifically, line 9 corresponds to using operator $Q_2(\boldsymbol{\sigma}_{ij}) = 2\mathcal{P}_2(\boldsymbol{\sigma}_{ij}) - \boldsymbol{\sigma}_{ij} = \tilde{\boldsymbol{\sigma}}_{ij}$, with $\mathcal{P}_2(\boldsymbol{\sigma}_{ji}) = (\boldsymbol{\sigma}_{ij} + \tilde{\boldsymbol{\sigma}}_{ij})/2$, and, analogously, $Q_2(\tilde{\boldsymbol{\sigma}}_{ji}) = \boldsymbol{\sigma}_{ji}$. This means that the projected offloading variables of node i, $\boldsymbol{\sigma}_{ij}$ (from $i \longrightarrow j$), must take the value of $\tilde{\boldsymbol{\sigma}}_{ij}$ (indicating, at neighbor j, what is taken from i). Lines 10-11 correspond to using Q_1, as well as extracting $\boldsymbol{x}_1^+$ according to (36). On line 14, the Anderson acceleration (A2DR in [32]) is applied, considering all the values of $\boldsymbol{z}_i$ from a circular buffer Z, of fixed length.

Algorithm 3 MPC based allocation of processing tasks

```
1:  Input: convergence parameters $\rho$ and $q$, stopping threshold $f_{\text{obj}}$ and $z_{\text{obj}}$, cost function $J$ (or $\tilde{J}$), buffer $Z$ of fixed length
2:  Output: workload allocation for the current step
3:  initialize: $\boldsymbol{z} = \boldsymbol{0}$, $\boldsymbol{x} = \boldsymbol{0}$, $f_{\text{cur}} = \infty$, $z_{\text{cur}} = \infty$
4:  while $f_{\text{cur}} > f_{\text{obj}}$ or $z_{\text{cur}} > z_{\text{obj}}$ do
5:      for all nodes $i$ in $\mathcal{N}$ do
6:          $\boldsymbol{z}_i^{\text{old}} \leftarrow \boldsymbol{z}_i$
7:          transmit entries $\boldsymbol{\sigma}_{ij}$ and $\tilde{\boldsymbol{\sigma}}_{ji}$ to neighbors $j \in \mathcal{N}_i$
8:          receive entries $\boldsymbol{\sigma}_{ji}$ and $\tilde{\boldsymbol{\sigma}}_{ij}$ from neighbors $j \in \mathcal{N}_i$
9:          $\boldsymbol{\sigma}_{ij} \leftarrow \tilde{\boldsymbol{\sigma}}_{ij}$ and $\tilde{\boldsymbol{\sigma}}_{ji} \leftarrow \boldsymbol{\sigma}_{ji}$        ▷ apply $Q_2$
10:         $\boldsymbol{x}_i \leftarrow \mathcal{P}_1(\boldsymbol{z}_i/\sqrt{\rho})$        ▷ solve problem (33)
11:         $\boldsymbol{z}_i \leftarrow 2\sqrt{\rho}\,\boldsymbol{x}_i - \boldsymbol{z}_i$        ▷ complete $Q_1$
12:         $\boldsymbol{z}_i \leftarrow (1-q)\boldsymbol{z}_i^{\text{old}} + q\boldsymbol{z}_i$        ▷ combine with $\boldsymbol{z}_i^{\text{old}}$
13:         add $\boldsymbol{z}_i$ to buffer $Z$
14:         $\boldsymbol{z}_i \leftarrow$ A2DR(Z)        ▷ Anderson acceleration [32]
15:     end for
16:     $f_{\text{cur}} \leftarrow J(\boldsymbol{x})$
17:     $z_{\text{cur}} \leftarrow \|\boldsymbol{z} - \boldsymbol{z}^{\text{old}}\|$
18: end while
19: locally allocate workload at slot $k = 1$ using $\boldsymbol{x}_i$
```

VI. Numerical Results

A. Simulation scenario and parameters

1) Scenario: As a reference scenario, we consider the three-tier network of Fig. 3, with tiers T_0, T_1 and T_2. At tier T_0, IoT nodes and other end devices generate workload according to the Markovian generation process described in Section III-D. For job arrivals, we consider bursts of length $b = \max(3, \epsilon/(1-\epsilon))$, with ϵ being the generation rate. Energy, arrives in bursts of fixed average length of 50 slots, following a correlated generation process with $\epsilon_e = 0.6$. Data that needs processing is sent to the closest access server in tier T_1, where the actual optimization takes place. The results that follows, have been obtained averaging over 10 randomly generated networks, each with $M = 16$ servers. Network connectivity graphs are obtained by independently generating two undirected and connected graphs with low average degree, one for T_1 servers, with 12 nodes, and the other one for T_2 servers, with 4 nodes. Moreover, T_1 servers have either a *directed* connection to one (and only one) server of layer T_2, with probability $p = 0.5$, or no connections to T_2 servers (prob. $1 - p$). Accordingly, workload can be only sent from T_1 to T_2 servers, but it cannot be sent back to tier T_1. Any computation resource can redirect data and workload to the cloud if the required processing cannot be performed on time by the edge server infrastructure.

The simulation parameters, listed in Tab. II, are typical of image processing tasks. We consider T_2 servers twice as powerful as T_1 servers, and with a double transmission rate as well. The harvested energy covers on average 30% of the maximum computation power of each server. Nodes are also equipped with a small energy buffer, that is kept above 25% of the maximum capacity, i.e., $e_{i,k} \geq \underline{b}_i = 0.25 \times \bar{b}_i$, $\forall\, k$, purchasing energy from the power grid if necessary, and selling it if the residual harvested energy exceeds $\bar{b}_i$.

2) Evaluation metrics: **Jain's fairness index:** To assess the load balancing capability of job scheduling solutions, we use the Jain's fairness index,

$$F(\phi) = \frac{\left(\sum_{i\in\mathcal{M}} \phi_i\right)^2}{|\mathcal{M}| \sum_{i\in\mathcal{M}} \phi_i^2}, \tag{37}$$

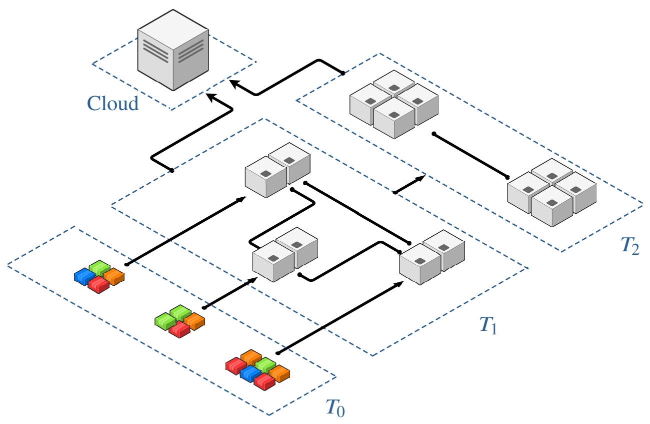


Fig. 3: **Three-tier network**. The three tiers T_0, T_1 and T_2 comprise devices generating workload (e.g., Internet of Things (IoT) nodes and mobile terminals), edge servers, and more powerful edge servers, respectively. Besides, edge servers can also fall back to the cloud computing facility.

TABLE II: Summary of simulation parameters

General parameters	
simulations length	1050 slots
transient discarded	50 slots
number of workload buffers D	6
state-control weight tradeoff γ	0.5
energy required per operation c	10 J/Gflop
energy required per transmitted Mbit to close servers m_{ij}	0.67 J/Mbit
energy required per transmitted Mbit to the cloud m_{ic}	$2m_{ij}$
operations required per Mbit of data op	0.33 Gflop/Mbit
Parameters for T_1 servers	
average harvested energy (when burst is active) h_1	530 J/slot
harvested energy std σ_{h_1}	150 J/slot
maximum computational power $\bar{w}_1$	100 Gflop/slot
maximum transmission rate $\bar{o}_1$	10 Gbit/slot
maximum amount of data processed per slot $\bar{d}_1$	300 Mbit/slot
maximum battery capacity $\bar{b}_1$	2×10^4 J
average incoming workload per buffer (when burst is active) μ_d	120 Mbit/slot
incoming workload std per buffer σ_{μ_d}	22 Mbit/slot
Parameters for T_2 servers	
average harvested energy (when burst is active) h_2	1000 J/slot
harvested energy std σ_{h_2}	190 J/slot
maximum computational power $\bar{w}_2$	$2\bar{w}_1$
maximum transmission rate $\bar{o}_2$	$2\bar{o}_1$
maximum amount of data processed per slot $\bar{d}_2$	$2\bar{d}_1$
maximum battery capacity $\bar{b}_2$	$2\bar{b}_1$

where $\phi = \{\phi_i\}_{i \in \mathcal{M}}$ collects the servers load factors, see (15), averaged across the whole simulation horizon.

System efficiency: we respectively define E_h and E_p as the total amount of energy harvested and the total energy purchased by the power grid. The ratio $\eta_e = E_h/(E_h + E_p)$ is a measure of energy efficiency. In fact, $\eta_e = 1$ if $E_p = 0$, i.e., if the system is operated by solely exploiting the energy harvested, whereas $\eta_e < 1$ if $E_p > 0$. With W_{edge}, we mean the total amount of workload processed by the edge servers in tiers T_1 and T_2, and with W_{cloud} we indicate the total amount of workload that is sent to the cloud computing facility. The ratio $\eta_w = \mu_1 W_{\text{edge}}/(\mu_1 W_{\text{edge}} + \mu_2 W_{\text{cloud}})$ weighs the capability of the computing infrastructure of handling all processes at the network edge and $\eta_w = 1$ only if $W_{\text{cloud}} = 0$. The scaling factors μ_1 and μ_2 translate the amount of workload into the associated energy consumption. As we also want to assess the fairness of the allocation, the total efficiency metric is defined as

$$\eta = \eta_e \times \eta_w \times F(\phi)\,. \tag{38}$$

For our results, we used $\mu_2 = 5\mu_1$, as the carbon footprint of cloud computing is usually higher than that of edge servers, mainly due to the energy hungry cooling systems that are used at the cloud. Note that $0 \leq \eta \leq 1$ and $\eta = 1$ only when the system solely uses harvested energy, executes all the tasks inside the edge network, and the workload is perfectly balanced across the edge servers.

Duty cycle: is the fraction of time during which a server is switched on. It is defined, for every server $i \in \mathcal{M}$ as the number of time slots τ_i in which the server is active, divided by the total number of slots T, namely,

$$D_i = \frac{\tau_i}{T}. \tag{39}$$

3) Low complexity heuristic: For benchmark purposes, we consider a simple and yet reasonable heuristic, as follows: i) edge servers execute workload locally in ascending order of their deadlines, without offloading data until the maximum computing capacity is reached. ii) If the amount of workload allocated to server i exceeds its computing capacity, it offloads part of such workload to the freest of its neighbors $j \in \mathcal{N}_i$, by offloading data to j until the workload difference at i and j is smallest (ideally zero). If, however, i itself is the freest server in $\{\mathcal{N}_i \cup i\}$, it will not offload anything. iii) Workload is sent from an edge server to the cloud facility only from buffer $d = 1$, and only if it is impossible to execute it on time at the edge server. iv) At each time slot, edge server i computes the local energy expenditure and trades energy with the power grid in such a way that its battery level is at least 25% of its battery capacity.

B. Performance analysis: optimal vs heuristic policies

The same evolution of job and energy arrivals (same models and parameters) was used to obtain the following plots, for all algorithms. Moreover, 95% confidence bands are shown as shaded areas surrounding the curves.

A preliminary performance analysis is presented in Fig. 4: the proposed distributed optimization framework of Algorithm 3 is indicated by "MPC ($N = x$)", where $x \in \{3, 8\}$ and N represents the length of the prediction window, "myopic" refers to the MPC framework with $N = 1$ and "heuristic" to the algorithm of Section VI-A3. The maximum job deadline is set to $D = 6$, and thus, with $N = 3$, MPC cannot predict the temporal evolution of those jobs with deadline greater than 3 time slots, while it can do so with $N = 8$ and, in general, with any $N \geq D$.

The average load of T_2 servers is shown in Fig. 4a. Although in our network scenario it is more difficult to fully exploit these servers because of the sparsity of the links, MPC correctly brings their utilization factor to 100% for an increasing load ($\epsilon \geq 0.7$). This is not attainable with the myopic and heuristic schemes. With the heuristic, the workload is executed as much as possible locally and, in turn, when ϵ is small only T_1 servers process jobs. Notably, the load factor at T_2 servers for the heuristic is lower than that of the myopic scheme even when

T_1 servers are full. This fact is connected with the results of Fig. 4d, which shows the amount of workload sent to the cloud computing facility. In particular, optimizing in a myopic way or employing a heuristic workload allocation policy leads to a much more intensive use of cloud computing resources starting from $\epsilon = 0.5$, when T_2 servers are only 40% full. This corresponds to a poor scheduling of the jobs, which should be ideally sent to the cloud facility only when all the edge servers are fully exploited.

On the other hand, optimizing in a predictive way, even with a small lookahead window, i.e., MPC with $N = 3$, brings the advantage of only using the cloud facility when the edge system operates at full capacity, i.e., beyond $\epsilon = 0.7$. As a consequence, since the carbon footprint of the cloud facility is higher than that of the edge network, the energy that is drained globally (edge and cloud), shown in Fig. 4e, is smaller for the proposed algorithm (see the range $\epsilon \in [0.5, 0.8]$). From Fig. 4b, we further see that the amount of energy harvested suffices to keep the battery level to 100% until $\epsilon = 0.4$, irrespectively of the used method. In this region, thus, not only the system is fully self-sufficient, but can also inject excess energy into the power grid. Beyond this load, the heuristic and MPC behave differently. At low values of ϵ (i.e., $\epsilon \leq 0.5$), MPC and the heuristic lead to high battery levels, as the energy harvested is sufficient to fully satisfy the computing demand. As ϵ increases beyond 0.5, MPC exploits the available computing resources in T_2, leading to a smaller energy reserve for these servers. Instead, the heuristic sends more workload to the cloud computing facility, under allocating T_2 servers.

The Jain's fairness index (37) is plotted in Fig. 4c. As can be seen, a prediction horizon of $N = 8$ leads to a good balancing of computing resources, maintaining the fairness index above 0.85 even at very low load. With the myopic scheme, the performance slightly degrades, dropping considerably in the range $\epsilon \in (0, 0.3]$. A further substantial drop is observed with the heuristic.

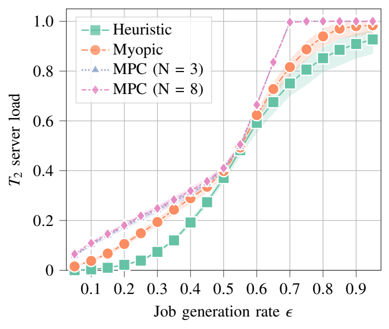
These facts directly reflect on the global system efficiency η (see Fig. 4f), which is very low for the proposed heuristic, across all values of ϵ. MPC's efficiency is highest at low values of ϵ, as it more effectively balances the load across the edge servers (Fig. 4c), and it remains highest as ϵ increases, as MPC sends less workload to the cloud facility (Fig. 4d).

Figs. 5a and 5b show the energy traded with the power grid (respectively, sold and purchased). MPC significantly reduces the amount of energy injected into the power grid with respect to myopic and heuristic strategies. At low ϵ, e.g., around $\epsilon = 0.3$, the amount of energy sold goes from about 1.6 kJ/slot of the heuristic policy to about 0.8 kJ/slot of MPC, see Fig. 5a. Also, MPC buys less energy from the power grid, going from 1 kJ/slot (heuristic) to 0.6 kJ/slot (MPC), see Fig. 5b. This reflects a more efficient management of harvested energy resources by MPC, resulting in a reduction of 50% in the energy traded with the grid. Beyond $\epsilon \approx 0.6$, MPC acquires more energy, as that coming from renewables is no longer sufficient to fully cope with the increased processing demand at the edge. Instead, myopic and heuristic schemes purchase less energy due to their poorer allocation of computing resources, and send more workload to the cloud facility (see Fig. 4d).

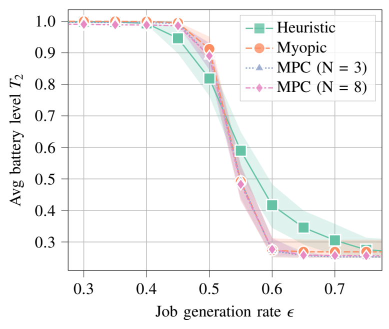
In Fig. 6, we analyze the impact of the predictor used for MPC. Specifically, the genie predictor is compared with that based on Markov chains, with known transition probabilities, and with an i.i.d. predictor, which uses the average intensity of the arrivals, see Section III-D. Fig. 6a shows the dependency between the amount of data sent to the cloud facility and the prediction horizon N. As expected, the genie predictor performs best, completely preventing the system from sending workload to the cloud starting from $N = 3$, whereas a higher N is required for the other (less accurate) predictors to achieve the same goal. This is motivated by the fact that the random samples used to obtain a predicted trajectory more accurately reveal the average (intensity) of the process as their number increases (higher N). Therefore, with a sufficiently long prediction window, even very simple predictors such as the i.i.d. one can be used profitably, as long as the average arrival rate is accurately estimated. In Fig. 6b, the Jain's fairness index is shown as a function of ϵ: both the Markov and the i.i.d. predictors lead to very similar performance, which is close to that of the genie, for any ϵ. The very good quality of these predictors is also confirmed by the system efficiency η (Fig. 6c): although the difference is negligible, there is a slight advantage in using Markov chains at low generation rates.

C. *Load balancing vs consolidation*

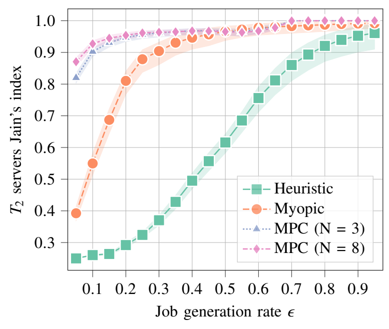
We now assess the role of the two cost functions of Section IV-D. The results are obtained setting $D = 6$, with job arrivals prevented in the two queues that are closest to the deadline, but increasing the average workload arrival rate to $\mu_d = 140$ Mbit/slot per queue, when the MC is in the ON state. This is motivated by the fact that jobs close to the deadline cannot be migrated, and therefore it would be difficult to highlight the consolidation aspect in their presence. From Fig. 7, we see that the quadratic cost promotes load balancing, while the logarithmic one and the heuristic scheme both induce server consolidation. Specifically, in Fig. 7a the fraction of active servers is plotted as a function of ϵ. At low generation rates, the non-convex (logarithmic) cost reduces the number of active servers with respect to the convex one by up to 40%. For $\epsilon < 0.3$, the proposed heuristic achieves the best results in terms of server consolidation. This holds true because, unlike optimal policies, it executes all the incoming workload at the edge server that receives it in the first place, without migrating it. Moreover, MPC may also send computing tasks to energy rich servers, going against the consolidation objective, and in the interest of exploiting as much as possible the available energy resources. In addition, as soon as the the arrival rate increases a bit, the heuristic produces an oscillatory behavior in the server activity status, by continuously switching on and off the edge servers. Instead, MPC avoids this undesirable ping ponging between activity statuses. Moreover, with MPC the servers that are kept off are consistently the same ones, and are selected based on their energy availability. Fig. 7b ($\epsilon = 0.25$) shows that the heuristic leads to an imbalance in the way the servers are exploited across the two tiers, whereas MPC achieves a more balanced allocation, setting similar duty cycles for the active servers in T_1 and T_2.



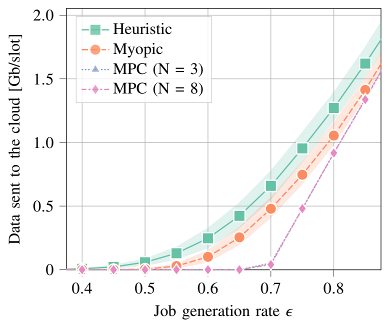
(a)



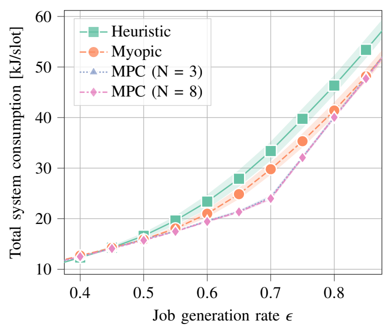
(b)



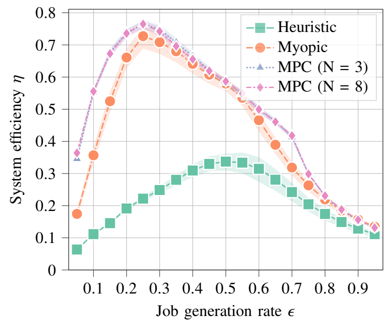
(c)



(d)

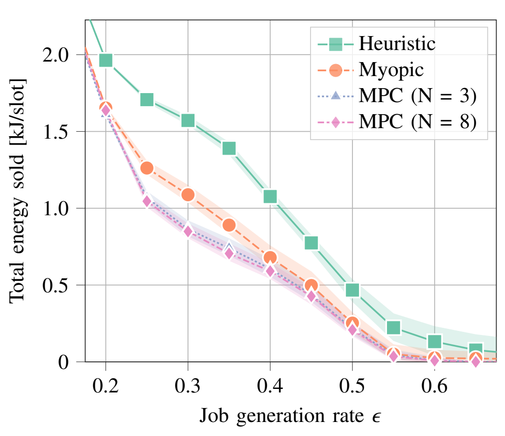


(e)

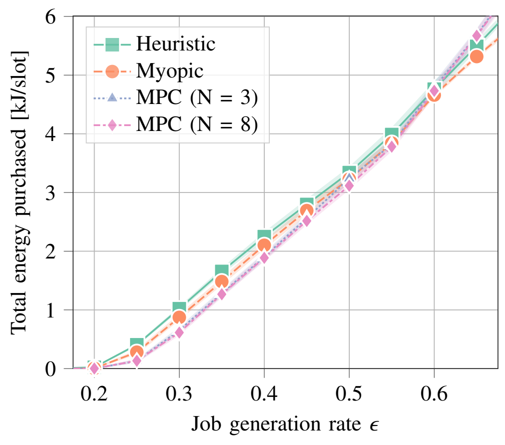


(f)

Fig. 4: Main system features as a function of the job generation rate. In these plots, the quadratic cost function is used for comparison with the benchmark heuristic.



(a) Sold



(b) Purchased

Fig. 5: Average energy traded with the power grid.

D. Convergence of the distributed MPC scheme

In Fig. 8, numerical insights on the convergence time of the distributed MPC solution (Algorithm 3) are given, for a network with $M = 16$ servers, $N = 8$, and $\epsilon = 0.5$. The numerical solver uses CVXPY [33] and OSQP [34] to process each local iteration of the DRS algorithm. The quadratic cost leads to a very fast convergence: the distance from the optimum gets smaller than 10^{-4} just after 15 iterations. For the logarithmic cost, since an optimal solver is not available, the optimal objective $f(x^*)$ is approximated as the value of $f(\cdot)$ obtained by the iterative solver after 1,000 iterations. In this case, a linear convergence is no longer achieved, the behavior of $|f(x)/f(x^*)|$ shown in Fig. 8 is non-monotonic, and convergence is slower.

This is also confirmed by Fig. 9, where the median number of iterations needed for convergence is shown as a function of ϵ (the shaded regions indicate the interquartile ranges). As a stopping criterion, we require that $z_{\text{obj}} = f_{\text{obj}} = 0.01$. For the quadratic cost, the convergence time remains about constant and with a small variance until $\epsilon \approx 0.6$, increasing at higher loads. The same median holds for the non-convex formulation, but in this case the interquartile range covers a wider area, denoting that, in this case, the solution is highly sensitive to initial conditions, and may require a higher number of iterations to converge. At very high loads (beyond $\epsilon = 0.8$), the

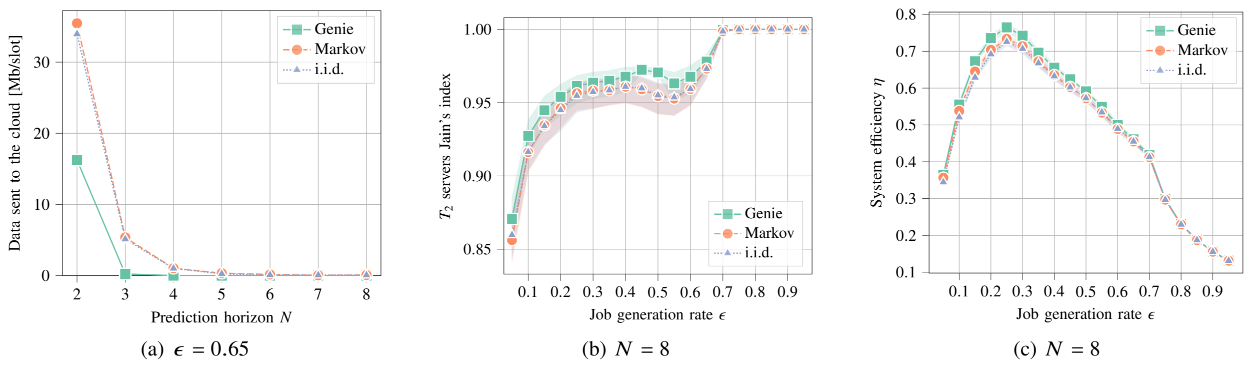


Fig. 6: Comparison between two simple predictors, namely a Markov chain and an i.i.d. predictor, with respect to the genie policy. They are used by the MPC based optimization scheme with the quadratic objective function.

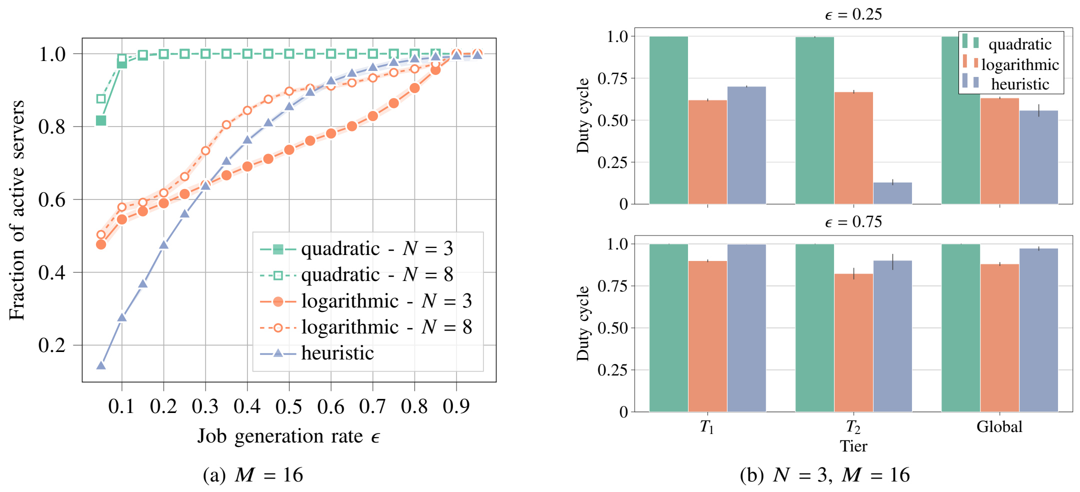


Fig. 7: Comparison of the proposed optimization schemes as per the consolidation metrics.

convergence time of the logarithmic cost increases abruptly. However, note that, in the main working region of the system, where all the workload can be processed by the edge servers, both methods require fewer than 50 iterations to converge. Furthermore, we observe that a consolidation approach (logarithmic cost) makes little sense at high load, say $\epsilon \geq 0.5$, where nearly all the servers are to be used anyway and, in turn, the convex cost represents a better choice. In fact, using the logarithmic cost, makes sense at low load, where the associated formulation converges quickly.

From these findings, we recommend using a convex (quadratic) cost at all ϵ if the objective is to promote load balancing across the servers, whereas if the aim is to promote consolidation, it makes sense to use a non-convex (logarithmic) cost until, e.g., $\epsilon \approx 0.5$, and use the convex one at higher loads. This is because, as the load increases, server consolidation becomes an ill posed objective, and the use of a logarithmic cost would only lead to a slower convergence, leading to the same solution attained by the quadratic cost formulation.

VII. Conclusions

In this paper, the problem of decreasing the energy drainage associated with processing tasks in MEC networks is tackled, considering edge servers equipped with batteries and energy harvesting devices. An online, predictive and fully decentralized optimization framework for the allocation of computing tasks is developed, exploiting MPC in conjunction with a customized version of the DRS algorithm. Two contrasting objectives, namely, load-balancing and consolidation, are sought. The results show that the proposed algorithm

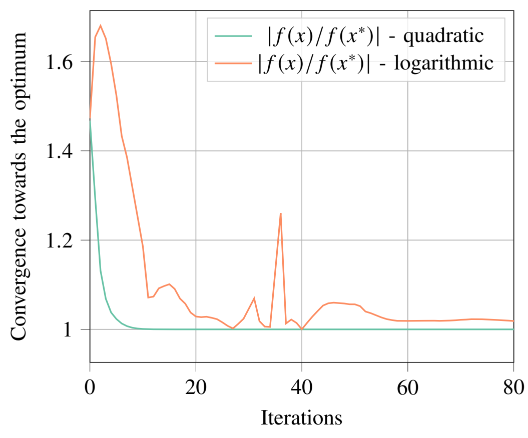


Fig. 8: DRS convergence to the optimal value for $\epsilon = 0.5$.

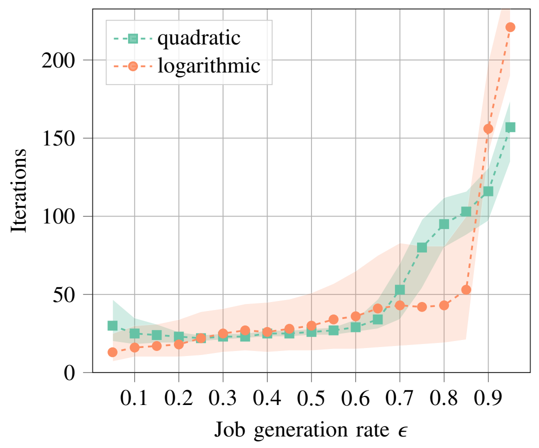


Fig. 9: Median number of iterations of DRS to reach convergence for $z_{\text{obj}} = f_{\text{obj}} = 10^{-2}$. The shaded regions represent the interquartile range. The prediction window is set to $N = 8$.

is beneficial with respect to a heuristic strategy, and to an approach based on myopic optimization. The resulting job scheduling algorithm uses the harvested energy much more effectively, by exploiting energy rich edge servers and reducing the amount of energy acquired from the power grid. When the consolidation objective is pursued, the fraction of active servers is reduced by up to 40%. Open research avenues are the study of workload allocation strategies by accounting for user mobility, and new energy consumption models for modern GPU based architectures.

Acknowledgment

This work has received funding from the Italian Ministry of Education, University and Research (MIUR) through the PRIN project no. 2017NS9FEY entitled "Realtime Control of 5G Wireless Networks: Taming the Complexity of Future Transmission and Computation Challenges", and has also been supported, in part, by MIUR through the initiative "Departments of Excellence" (Law 232/2016). The views and opinions expressed in this work are those of the authors and do not necessarily reflect those of the funding institutions.

References

[1] D. Reinsel, J. Gantz, and J. Rydning, "The digitization of the world from edge to core," International Data Corporation (IDC), Tech. Rep., 2018, https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

[2] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, 2017.

[3] W. Li, T. Yang, F. C. Delicato, P. F. Pires, Z. Tari, S. U. Khan, and A. Y. Zomaya, "On enabling sustainable edge computing with renewable energy resources," *IEEE Communications Magazine*, vol. 56, no. 5, 2018.

[4] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, WI, 2017, vol. 2.

[5] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, 2011.

[6] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, 2017.

[7] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, 2016.

[8] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, 2017.

[9] W. John, C. Sargor, R. Szabo, A. J. Awan, C. Padala, E. Drake, M. Julien, and M. Opsenica, "The future of cloud computing: highly distributed with heterogeneous hardware," Ericsson, Tech. Rep., 2020, https://www.ericsson.com/en/reports-and-papers/ericsson-technology-review/articles/the-future-of-cloud-computing.

[10] J. Xu, L. Chen, and S. Ren, "Online Learning for Offloading and Autoscaling in Energy Harvesting Mobile Edge Computing," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 3, 2017.

[11] M. Berno, J. J. Alcaraz, and M. Rossi, "On the Allocation of Computing Tasks under QoS Constraints in Hierarchical MEC Architectures," in *Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, 2019.

[12] L. A. Barroso and U. Hölzle, "The Case for Energy-Proportional Computing," *Computer*, vol. 40, no. 12, 2007.

[13] R. Sen and D. A. Wood, "Energy-Proportional Computing: A New Definition," *Computer*, vol. 50, no. 8, 2017.

[14] B. Subramaniam and W. Feng, "Towards Energy-Proportional Computing for Enterprise-Class Server Workloads," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. Association for Computing Machinery, 2013.

[15] A. Al-Shuwaili and O. Simeone, "Energy-Efficient Resource Allocation for Mobile Edge Computing-Based Augmented Reality Applications," *IEEE Wireless Communications Letters*, vol. 6, no. 3, 2017.

[16] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic Computation Offloading for Mobile-Edge Computing With Energy Harvesting Devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, 2016.

[17] A. Bozorgchenani, S. Disabato, D. Tarchi, and M. Roveri, "An energy harvesting solution for computation offloading in Fog Computing networks," *Computer Communications*, vol. 160, 2020.

[18] B. Li, Z. Fei, J. Shen, X. Jiang, and X. Zhong, "Dynamic Offloading for Energy Harvesting Mobile Edge Computing: Architecture, Case Studies, and Future Directions," *IEEE Access*, vol. 7, 2019.

[19] H. Wu, L. Chen, C. Shen, W. Wen, and J. Xu, "Online Geographical Load Balancing for Energy-Harvesting Mobile Edge Computing," in *IEEE International Conference on Communications (ICC)*, 2018.

[20] D. Cecchinato, M. Berno, F. Esposito, and M. Rossi, "Allocation of computing tasks in distributed mec servers co-powered by renewable sources and the power grid," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020.

[21] M. Gaggero and L. Caviglione, "Predictive Control for Energy-Aware Consolidation in Cloud Datacenters," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 2, 2016.

[22] L. Wang, J. Xu, H. A. Duran-Limon, and M. Zhao, "QoS-Driven Cloud Resource Management through Fuzzy Model Predictive Control," in *IEEE International Conference on Autonomic Computing*, 2015.

[23] M. Rahmani-andebili and H. Shen, "Cooperative distributed energy scheduling for smart homes applying stochastic model predictive control," in *IEEE International Conference on Communications (ICC)*, 2017.

[24] P. Skarin, W. Tarneberg, K. Arzen, and M. Kihl, "Towards Mission-Critical Control at the Edge and Over 5G," in *IEEE International Conference on Edge Computing (EDGE)*, 2018.

[25] P. Giselsson and S. Boyd, "Linear convergence and metric selection for Douglas-Rachford splitting and ADMM," *IEEE Transactions on Automatic Control*, vol. 62, no. 2, 2016.

[26] T. Erseghe, "New results on the local linear convergence of ADMM: a joint approach," *IEEE Transactions on Automatic Control*, 2020.

[27] A. Makhdoumi and A. Ozdaglar, "Convergence Rate of Distributed ADMM Over Networks," *IEEE Transactions on Automatic Control*, vol. 62, no. 10, 2017.

[28] J. F. C. Mota, J. M. F. Xavier, P. M. Q. Aguiar, and M. Puschel, "Distributed Optimization With Local Domains: Applications in MPC and Network Flows," *IEEE Transactions on Automatic Control*, vol. 60, no. 7, 2015.

[29] T. Sun, P. Yin, L. Cheng, and H. Jiang, "Alternating direction method of multipliers with difference of convex functions," *Advances in Computational Mathematics*, vol. 44, no. 3, 2018.

[30] I. Necoara, V. Nedelcu, and I. Dumitrache, "Parallel and distributed optimization methods for estimation and control in networks," *Journal of Process Control*, vol. 21, no. 5, 2011.

[31] F. Farokhi, I. Shames, and K. H. Johansson, "Distributed MPC via dual decomposition and alternative direction method of multipliers," in *Distributed model predictive control made easy*. Springer, 2014.

[32] A. Fu, J. Zhang, and S. Boyd, "Anderson accelerated Douglas-Rachford splitting," *SIAM Journal on Scientific Computing*, vol. 42, no. 6, pp. A3560–A3583, 2020.

[33] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, 2016.

[34] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, 2020.

Giovanni Perin (S'19) received the B.Sc. degree in Information Engineering and the M.Sc. degree in ICT for Internet and Multimedia (summa cum laude) from the University of Padova (Italy) in 2017 and 2019, respectively. In 2019, he spent six months as a visiting student at the Deutsche Telekom Chair of Communication Networks, Technical University of Dresden (Germany), working on broadcast routing. Since 2019, he has been enrolled in the Ph.D. program in Information Engineering at the University of Padova, under the supervision of Prof. Michele Rossi, joining the PRIN project (n. 2017NS9FEY) tackling the real-time control of 5G wireless networks. His research focuses on sustainable edge computing, distributed optimization and processing, and federated learning.

Michele Berno received the B.Sc. degree in Information Engineering and the M.Sc. degree (Hons.) in Telecommunications Engineering from the University of Padova, Italy, in 2015 and 2017, respectively. He currently is a Ph.D. candidate in ICT Engineering with the SIGNET research group in the Department of Information Engineering (University of Padova). Since 2019, he is a member of the PRIN project (n. 2017NS9FEY) tackling the real-time control of 5G wireless networks. During 2019, he was a research scholar with the Department of Computer Science, Saint Louis University (SLU), Missouri, US. He was a recipient of two best paper awards (FMEC 2019, IEEE MobileCloud 2020). His research interests include sustainable edge/cloud computing, distributed optimization, and machine learning.

Tomaso Erseghe is an Associate Professor at the University of Padova, Italy. He received a Laurea (M.Sc. degree) and a Ph.D. in Telecommunication Engineering from the University of Padova, Italy in 1996 and 2002, respectively. From 1997 to 1999 he was with Snell & Wilcox, an English broadcast manufacturer. From 2003 to 2017 he was an Assistant Professor (Ricercatore) at the Department of Information Engineering, University of Padova. His research interests have covered the fields of coding in the finite block-length regime, social network analysis and network science, distributed algorithms, smart grid optimisation, ultra-wideband transmission systems design, spectral analysis of complex modulation formats, fractional Fourier transforms and their applications, image processing and compression.

Michele Rossi (SM'13) is a Full Professor of Telecommunications in the Department of Information Engineering (DEI) at the University of Padova (UNIPD), Italy, teaching courses within the Master's Degrees in ICT for internet and Multimedia at DEI (http://mime.dei.unipd.it/ and Data Science, offered by the Department of Mathematics (DM) at UNIPD (https://datascience.math.unipd.it/). Since 2017, he has been the Director of the DEI/IEEE Summer School of Information Engineering (http://ssie.dei.unipd.it/). His research interests lie in wireless sensing systems, green mobile networks, edge and wearable computing. In recent years, he has been involved in several EU projects on IoT technology (e.g., IOT-A, project no. 257521), and has collaborated with companies such as DOCOMO (compressive dissemination and network coding for distributed wireless networks) and Worldsensing (optimized IoT solutions for smart cities). In 2014, he was the recipient of a SAMSUNG GRO award with a project entitled "Boosting Efficiency in Biometric Signal Processing for Smart Wearable Devices". In 2016-2018, he has been involved in the design of IoT protocols exploiting cognition and machine learning, as part of INTEL's Strategic Research Alliance (ISRA) R&D program. His research is currently supported by the European Commission through the H2020 projects SCAVENGE (no. 675891) on "green 5G networks", MINTS (no. 861222) on "mm-wave networking and sensing" and GREENEDGE (no. 953775) on "green edge computing for mobile networks" (project coordinator). Dr. Rossi has been the recipient of seven best paper awards from the IEEE and currently serves on the Editorial Boards of the IEEE Transactions on Mobile Computing, and of the Open Journal of the Communications Society.