UNIVERSITY OF PADOVA

DEPARTMENT OF INFORMATION ENGINEERING
Ph.D. Course in Information Engineering
Information and Communication Technologies Curriculum
XXXV series

Optimizing edge computing resources towards greener networks and services

Ph.D. Candidate
Giovanni Perin

Ph.D. Supervisor
Professor Michele Rossi

Ph.D. Co-supervisor
Professor Tomaso Erseghe

Ph.D. Coordinator
Professor Andrea Neviani

Academic Year
2021–2022

To my mom and dad, to whom
I owe everything I am today.

Abstract

The world is at the dawn of a new era, characterized by a large number of connected devices and a resulting massive availability of networked data. In this scenario, Multi-access Edge Computing (MEC) is the candidate reference paradigm to provide mobile users with low-latency processing and storage services. In contrast with Mobile Cloud Computing (MCC), MEC entails the deployment of computing facilities closer to the end devices, thus becoming a key enabler for applications such as augmented reality, tactile Internet, smart home, healthcare monitoring, connected cars, online gaming, etc. However, the problem of the energy efficiency of the decentralized MEC infrastructure arises. In this doctoral thesis, the management of the MEC platform is optimized to reduce the global carbon footprint of the network, i.e., to maximize the use of Renewable Energy Resources (RERs) for the initial placement and the subsequent execution and offloading of jobs. The main body of the thesis is organized into three chapters: the first one tackles the green energy management of edge servers equipped with a battery in a hierarchical MEC network and the electricity trade with the power grid; the second chapter considers a vehicular scenario where vehicles' trajectories are proactively tracked when migrating the users' computing tasks towards increasing the energetic efficiency of this process; the third chapter presents a comparison of the two decentralized optimization approaches designed, based on message passing. The results show that the proposed optimization frameworks, based on Model Predictive Control (MPC), can significantly reduce the carbon footprint of the edge network when compared to simple heuristics and other approaches in the scientific literature. The designed algorithms can reach almost complete carbon neutrality in a vast range of network conditions.

Contents

List of figures

List of tables

List of acronyms

Symbols

5G fifth-generation

A

ADMM Alternating Direction Method of Multipliers

AR Augmented Reality

B

BS Base Station

C

CAGR Compound Annual Growth Rate

CPU Central Processing Unit

D

D-MPC Distributed MPC

DL Distributed Learning

DRS Douglas-Rachford Splitting

E

EASE Energy-Aware job Scheduling at the Edge

EH Energy Harvesting

eNB evolved Node B

ETSI European Telecommunications Standards Institute

I

ICT Information and Communication Technologies

ID Identifier

IDC International Data Corporation

IoT Internet of Things

IoV Internet of Vehicles

IT Information Technology

M

M2M Machine-to-Machine

MC Markov Chain

MCC Mobile Cloud Computing

MDP Markov Decision Process

MEC Multi-access Edge Computing

MEH Mobile Edge Host

MIP Mixed Integer Programming

ML Machine Learning

MPC Model Predictive Control

P

PV Photovoltaic Panel

Q

QoS Quality of Service

QP Quadratic Programming

R

RER Renewable Energy Resource

RNN Recurrent Neural Network

S

SCA Successive Convex Approximation

SDG Sustainable Development Goal

U

UAV Unmanned Aerial Vehicle

UE User Equipment

UNO United Nations Organization

V

VM Virtual Machine

1

Introduction

Mobile Cloud Computing (MCC) has traditionally been seen as the main computing and storage facility provider for the Internet users. In recent years, however, we have witnessed a paradigm shift towards Multi-access Edge Computing (MEC) [1], [2]. MEC, by deploying the computing devices directly at the edge of the access networks, will significantly reduce the end-to-end delay between servers and clients, thus being the key enabler of applications that require extremely low latencies. A few examples of these are Augmented Reality (AR), healthcare monitoring, online gaming, remote control of robots, drones, and unmanned (aerial, water and ground) vehicles. To these, we can add all those applications that require processing data collected by sensors spread in the environment. The world is also facing the digital revolution of connected devices, which will increase dramatically the Internet traffic and data volume in the near future. In 2020, Cisco predicted that, by 2023, there will be more than 9 connected devices per capita in Europe and Machine-to-Machine (M2M) communications will be responsible for 50% of the traffic share. Among these, the vertical expected to be developing the most is connected cars, with a Compound Annual Growth Rate (CAGR) of 30% [3]. Equinix, a leading data center and edge computing company, forecasts that 800 billion USD will be spent on new edge equipment worldwide by 2028, with the COVID-19 pandemic most likely accelerating the transition to a digitized society [4].

This thesis analyzes the energy problem that arises in Information and Communication Technologies (ICT) and, specifically, in the modern MEC platform, proposing efficient solutions for the management and operation of the network infrastructure by integrating the use of Renewable Energy Resources (RERs) from the design phase.

1.1 The energy problem

As discussed above, the Internet traffic demand and the number of connected devices has experienced a large growth in recent years and is expected to maintain this trend for the near future. As an example, video content was responsible for 80% of the traffic share in 2020 and the work in [5] shows the different agents responsible for the energy consumption, from service generation at remote servers to the execution at end terminals. The cellular and access networks have the largest share of energy consumption (9500 and 4500 GWh, respectively), but metro and core networks also contribute significantly to the carbon footprint (2000 GWh). In [6] it is said that the streaming of an HD quality video for 10 minutes is comparable to a 2 kW oven working at full power for 5 minutes. MEC reduces the distance that the data has to travel across the network from the servers to the clients (i.e., avoiding metro and core networks) therefore inherently lowering energy consumption. Nonetheless, the management of a distributed platform has more complex challenges to be tackled concerning the paradigm of MCC, and using efficiently the energy can be a trickier task. Moreover, even though electronic devices are becoming more efficient, their deployment in communication networks will increase proportionally to the data traffic. As a direct consequence of the Jevons' paradox [7], which states that the increased efficiency of a resource and the consequent reduction of its cost will increase its demand (often with a higher rate), the global energy consumption of the ICT infrastructure is expected to increase significantly.

A possible (partial) solution to this challenge is the use of Energy Harvesting (EH) devices (e.g., Photovoltaic Panel (PV) panels) at the edge, as they can be easily equipped to the small servers spread in the urban environment. This proposal also supports one of the main outcomes of the Paris Agreement of 2015: the Sustainable Development Goals (SDGs) towards 2030, promoted by the United Nations Organization (UNO) [8]. Among these, objective nr. 7 seeks the use of "affordable and clean energy", incentivizing the use of renewable resources in the coming years. Furthermore, developing the MEC platform and enabling the previously mentioned verticals for both private customers and industry 4.0 directly involves objectives nr. 8 ("decent work and economic growth") and nr. 9 ("industry, innovation and infrastructures") of the SDGs.

In this thesis, the focus is on the design of algorithms for the management of edge computing facilities. The primary objective is the minimization of the carbon footprint of the infrastructure, i.e., finding the best ways to handle the energy coming from renewables to allocate computing and communications resources. An example of the system considered is depicted in Fig. 1.1.

1.2 Green solutions for jobs offloading in the MEC platform

This section introduces a review of the state-of-the-art algorithms to manage the task of jobs offloading in the MEC with the primary objective of reducing the energy consumption and/or maximizing the use of green sources, the motivation behind this thesis [11]. A specific related work section for the two main contributions of this thesis is provided in chapters 2 and 3.

To reduce the energy consumption of end devices, tasks can be partially or totally offloaded to

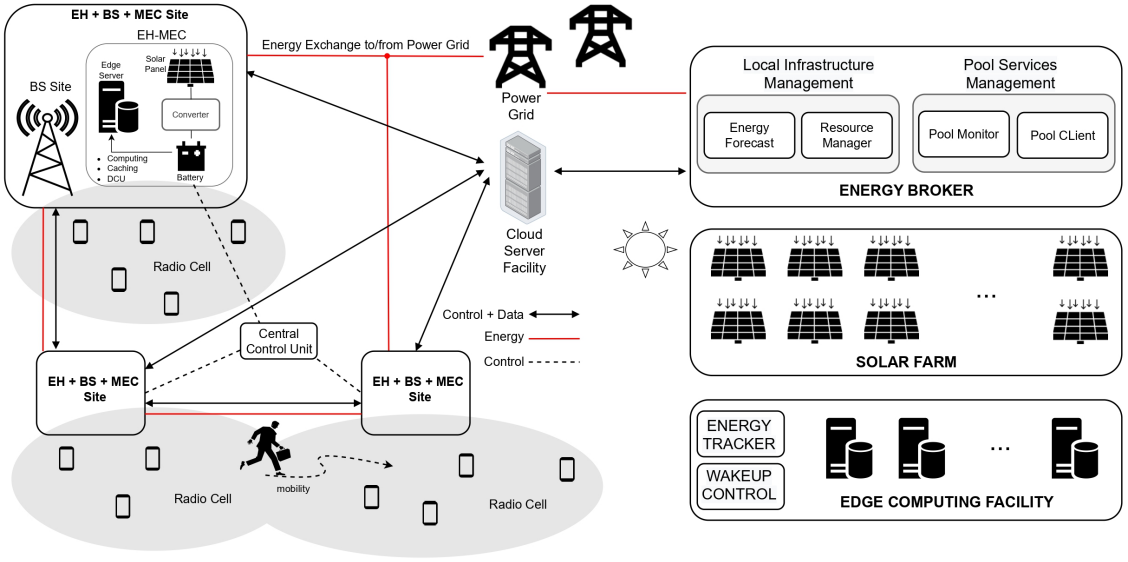


Figure 1.1: The considered architecture. A network of sustainable MEC cells, with BSs equipped with EH hardware, a battery, and an edge server is reported on the left. Edge computing resources can either be managed in a centralized or distributed [9] manner. A novel conceptual framework for edge computing through distributed renewable energy farms, edge computing facilities, and energy brokers to intelligently allocate computing resources across edge and cloud facilities is shown on the right [10].

the edge servers. This helps reduce the computation burden on end devices, which usually have limited processing and energy resources, while meeting processing deadlines. In some cases, end devices can execute a portion of the tasks.

Centralized schedulers The authors of [12] propose a constrained mixed-integer non-linear constrained program to jointly set the task offloading ratio (i.e., the fraction of task that is offloaded to the edge server(s)), transmission, and computing parameters subject to latency requirements, processing power, and memory availability. The problem is solved via a meta-heuristic algorithm that seeks to decrease the overall energy consumption across end devices and edge servers by leveraging the end devices' resources to their maximum extent. It is assumed that tasks can be decomposed into independent parts which can be executed in parallel and partially offloaded. The solution provides good schedules, achieving better energy savings with respect to using other meta-heuristic solvers. The processing energy consumption per Central Processing Unit (CPU) cycle is estimated as κf_{cpu}^2, where κ is a hardware-dependent constant (effective switched capacitance) and f_{cpu} the CPU frequency. The obtained policies achieve the desired behavior for a small/medium number of end devices. Furthermore, additional methods should be devised to distribute the load, to keep the complexity low in a distributed setup with many end nodes. In addition to optimizing the offloading ratio, a wise selection of the server(s), i.e., where to offload the task (or portion thereof), can further reduce energy consumption. The authors of [13] pro-

pose an end-to-end deep reinforcement learning (DRL) algorithm to maximize the number of executed tasks that are completed on time while minimizing the associated energy consumption. This method assumes monolithic tasks (no partitioning allowed) and leaves the study of tasks that can be partitioned as future work. Thanks to the use of DRL, this algorithm does not require modeling the MEC system dynamics (model-free optimization), which are learned sequentially. The algorithm reduces the energy cost of executed tasks through optimal server selection and computing power allocation for the task offloading. It uses the same energy model of [12] and may have scalability problems due to the centralized nature of the DRL solver. The authors of [14] propose a centralized intelligent energy management system to control energy consumption and flow in a two-tier mobile cellular network. A deep reinforcement learning algorithm is devised to maximize the use of the green energy harvested by observing and interacting with the surrounding environment. The algorithm objective amounts to reducing the energy drained from the power grid, subject to traffic level constraints. The environment is initially described through the harvested energy from each Base Station (BS), the BS battery levels, and the load experienced by the BSs across all hours of the day. The impact of different environment representations on network performance is then investigated. Numerical results indicate that considering the battery level and hour of the day is sufficient to achieve high performance.

The main difference with the works presented in this thesis is the development of decentralized network controllers, which better adapt to the distributed nature of devices in edge computing systems. This type of controller is presented in the following paragraph.

Distributed schedulers In the work [9], further described in Chapter 2, we propose a fully decentralized optimization framework based on Douglas-Rachford Splitting (DRS) to allocate the computing resources of edge servers and decide where to offload tasks subjected to a time deadline. The dynamic and multi-variable heuristic optimization algorithm distributes workload among MEC servers to achieve load balancing or server consolidation while encouraging RER utilization and reducing the transmission costs. The solution, model-based, employs Model Predictive Control (MPC), showing high performance even with simple predictors, e.g., just knowing the average value of the incoming processing load and energy from renewable resources. The algorithm's low complexity and fast convergent nature allow energy-efficient utilization of EH-MEC facilities. Numerical results show a 50% reduction in the RER energy that is sold to the power grid, which is instead being retained within the network to handle computing tasks. Furthermore, we conclude that consolidation is particularly beneficial when the network load is low, while load balancing is preferred in high load conditions. Overall, utilizing energy-aware, fast response, and low complexity schedulers results in an increased equivalent computation capacity for the edge network as a whole for a given energy consumption level. The elastic and predictive online scheduling algorithm for energy harvesting Internet of Things (IoT) edge networks in [15] demonstrates the benefits that are attainable through careful energy- and RERs-aware redistribution of the processing load. Their MPC based scheduler solves a sequence of low complexity convex problems, taking as input estimates for future processing and renewable energy arrivals and, based on these, outputs scheduling decisions using a look-ahead approach. This

method outperforms a recently proposed scheduling algorithm based on Markov Decision Processs (MDPs) [16] in terms of drop rate and residual energy reserve at the MEC nodes. Moreover, its performance is 90% of that of the globally optimal scheduler, obtained by an offline solver with perfect knowledge about future jobs and energy arrivals. Along the same lines, [17] investigates energy scheduling solutions to maximize green energy utilization on microgrid-connected MEC networks. Green energy availability and load demand arrivals are estimated. The optimization objective amounts to minimizing the residual expected energy and demand by capturing the tail risk of the uncertain energy and demand arrivals. This risk is used as the input metric for a distributed reinforcement learning mechanism to control the distributed computing resources. The proposed scheme is model-free and learns computing policies in an online fashion achieving up to 96% accurate energy scheduling.

1.2.1 Mobility-aware schedulers

Vehicular edge computing networks In the presence of user mobility, computing processes may be migrated among servers for a better user experience. If a mobile user generates a task prior to performing a handover to a different BS, the task may be migrated to the future serving BS, executed there, and delivered by the new serving BS to the end user following movement to the new radio cell. In making these decisions, several considerations arise. If the current server (before the handover) chooses to execute the task without offloading it, it may degrade the user's experience. If it decides to migrate the task, it must first choose *where* to offload it. Migration of the corresponding container/Virtual Machine (VM) must also occur, so that the destination server can continue the computation without interruption. In addition, deciding where to migrate the task further complicates the resource scheduling problem, as user mobility patterns also come into play. The authors of [18] devised an online algorithm to estimate the next cell hosting the mobile/vehicle user in a 5*G* vehicular network 4 seconds before the handover execution, with an accuracy of 88%. This method combines Markov Chains (MCs) and Recurrent Neural Network (RNN) based predictors, using MCs to forecast the mobility across BSs and RNNs for the fine-grained mobility tracking inside each BS site. Mobility estimation algorithms of this type are key to devising predictive schedulers to proactively allocate resources where they are actually needed. In [19], the work presented in Chapter 3, a task scheduling algorithm for EH-MEC networks serving vehicular users is proposed. The objective is to minimize the network carbon footprint subject to task latency constraints and mobility patterns. A centralized MPC-based algorithm is formulated as a first step to predict available green energy and computing resources based on workload arrivals at the Mobile Edge Hosts (MEHs) and handover probabilities in upcoming time slots. The proposed scheduler prioritizes executing those tasks with high computational intensities and those approaching deadlines, exploiting the available green energy. As a result, it prevents high task migration costs and packet loss. In a second phase, a distributed heuristic determines the best target servers for task migration by utilizing the trajectory predictor from [18]. As the objective is to minimize the energy consumption from the power grid, servers accept incoming tasks only if the locally available green energy is deemed sufficient to process them locally in future time slots.

The significant advancement of this thesis concerning [18] is that previous work does not consider the actual feasibility of the migration of the tasks in terms of computation capacities of the target host. In Chapter 3 we devise an algorithm to manage the network in an energy-efficient way while considering both delay and computational constraints.

UAV-assisted edge computing networks Infrastructure-assisted Unmanned Aerial Vehicle (UAV) systems are another remarkable example of mobility-aware edge networks. There, UAVs may be assigned to execute tasks in predefined positions such as monitoring/surveillance, network coverage extension, data traffic relay, and network assistance in high-demand hours. Since UAVs have often limited processing power and battery capacities, computing the desired tasks onboard may take longer than with a standard network server, which may result in excessive energy consumption. To overcome this, UAVs may offload their tasks to high processing power edge servers to reduce the decision-making time and save onboard energy. It is expected that a tradeoff between job offloading, flight time, and Quality of Service (QoS) requirements exists. The authors of [20] proposed a decision process where the UAV reads the network state and estimates the task completion time. It then solves an optimization problem to minimize a weighted sum of delay and energy consumption and decides whether to offload the task or not.

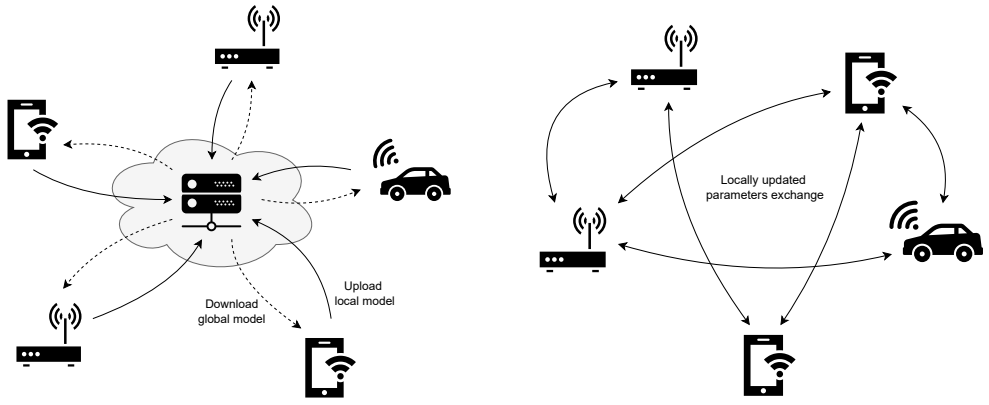
1.2.2 Distributed learning at the edge

Massive datasets and spare computational power are abundant at the network edge. Collaborative edge learning algorithms allow leveraging edge computational powers to train Machine Learning (ML) models locally, using the data collected by end nodes. Specifically, learning distributedly a model over N edge devices with global parameters $\boldsymbol{w}$ and dataset $\mathcal{D}$ amounts to solving the problem

$$\min \left[F\left(\boldsymbol{w} \mid \mathcal{D}\right) = \sum_{i=1}^{N} p_i F_i(\boldsymbol{w}_i \mid \mathcal{D}_i) \right], \tag{1.1}$$

for a set of weights p_i, where F_i are the local *loss functions* and $\boldsymbol{w}_i$ and $\mathcal{D}_i$ are the local model parameters and datasets, respectively.

Distributed ML models are appealing due to privacy considerations, as the local data that is collected at the distributed node sites does not have to be disclosed (sent). Local computations are instead performed at the nodes using their own data, and only model updates (gradients) are sent across the network. Although this is highly attractive, it also presents challenges related to the time-varying nature of computational power, communication rate, and energy availability at each device. Furthermore, communication, energy, and computation resources are often highly heterogeneous across wireless and edge devices, which leads to further complications. Hereafter, we refer to the difference in communication/computing/energy resources across nodes as *system heterogeneity*, while the fact that the data may be non-i.i.d. across devices is referred to as *statistical heterogeneity*. Utilizing heterogeneous edge resources to accurately train ML models in an energy-efficient manner is a challenging and still open endeavor. Distributed Learning (DL) algorithms come in two primary flavors: (i) *Federated Learning* (FL) [21], where the workers are con-



(a) Federated learning **(b) Fully decentralized learning**

Figure 1.2: Decentralized learning paradigms. On the left, a network of edge devices acting as clients locally train the model with their own data and periodically upload it to the server. The server aggregates the local models coming from the clients and sends to them an updated version of the global model. On the right, edge devices are connected via a general mesh topology. They update the model parameters using some message passing algorithm, by periodically communicating with their neighbor nodes and without involving any central aggregator.

nected with a star topology to a physical unit serving as an aggregator for their local models (see Fig. 1.2a), or (ii) *fully decentralized* communication/learning process, where workers share their locally updated ML model with their direct neighbors in any generic mesh topology (Fig. 1.2b), progressively reaching consensus on a common model.

Context-agnostic efficient communication

The amount of data exchanged for the model updates represents one of the major DL challenges (the communication bottleneck problem) due to the large size of typical ML models. This affects both the convergence time and the energy drained to transmit the model updates. According to [22], context-agnostic approaches to tackle this problem subdivide into (i) *sparsification*, i.e., pruning of some elements of the local gradients, (ii) *quantization*, i.e., reducing the number of bits to encode the local gradients, and (iii) *local Stochastic Gradient Descent* (*SGD*), namely, performing multiple local gradient descent steps before performing the aggregation. The first two methods reduce the number of bits sent for the aggregation steps, while the latter aims to reduce the total number of aggregation rounds needed.

Context-aware DL optimization

Despite the fact that the mechanisms mentioned above can be combined, this is not enough to make communication efficient in large-scale DL setups. In fact, the above strategies are independent of the communication channel and protocols used. A higher level of context awareness (in this case, channel characteristics and communication resources) in wireless networks holds the potential to significantly improve the efficiency of model training, as the channel quality varies

across network devices and over time, and local resources such as bandwidth and transmission power are limited. Scheduling and resource allocation thus assume a prominent role in improving the efficiency of DL.

Client selection and resources allocation *Client selection* refers to picking, at each learning step, a subset of clients that contribute to updating the model parameters within the global aggregation phase, hence saving network resources. As a simple approach, scheduling algorithms could select those clients with the best experienced channel. This choice, although speeding up the model training by avoiding the so-called *stragglers*, may result in a biased ML model in the presence of statistical heterogeneity, polarizing it towards the local models of those clients with a better channel quality, e.g., better positioned with respect to the access point. To cope with this, the authors of [22] propose a scheduling policy that considers the age of information (AoI) (i.e., the number of iterations from the last update) of each client in conjunction with the experienced channel quality. By penalizing this metric, they obtain a better performing model by using network (communication) resources efficiently. The use of analog transmissions to aggregate the model over-the-air is proposed in [23], where client selection is performed at the beginning of each round by optimizing (with a Lyapunov approach) a learning bound obtained in the paper, subject to communication and processing energy constraints. This procedure is shown to perform close to an optimal benchmark in terms of accuracy while outperforming a myopic scheduler. In papers [24]–[26] a convergence result is exploited to bound the squared norm of the gradients of the local model loss functions as

$$\left\|\nabla F_i\left(\boldsymbol{w}_i^k \mid \mathcal{D}_i\right)\right\|^2 \leq \theta \left\|\nabla F_i\left(\boldsymbol{w}_i^{k-1} \mid \mathcal{D}_i\right)\right\|^2, \tag{1.2}$$

for some $0 \leq \theta \leq 1$. In particular, when $\theta = 0$, the model will get an exact solution to the local problem, while if $\theta = 1$, no progress is observed between iterations $k-1$ and k. If the objective is strongly convex, the convergence rate is proven to be $\mathcal{O}(\log(1/\theta))$. In [24], the joint optimization of a weighted average of computing, transmission energy, and training time is considered. In their scenario, the server can select the proper bandwidth to communicate with the clients to adjust transmission energy and time while the clients decide their processing frequency locally. The authors of [25] realized a customized frequency division multiple access (FDMA) strategy to jointly optimize the energy used for communication and local computations by selecting the CPU frequency, the transmission power, and bandwidth and also providing a formal convergence analysis of the proposed algorithm. Their scheme outperforms selected benchmarks, including a time division multiple access (TDMA) approach, when looking at the tradeoff between energy and completion time. Further, in [26] a vehicular scenario is considered, where the authors perform both vehicle selection and resource allocation, again optimizing completion time and energy. The selection task is performed in a greedy manner to get higher image quality data, while the resource allocation is formulated as a min-max problem, devoted to minimizing the time and energy needed for the worst scheduled vehicle. Due to the absence of benchmark algorithms for a vehicular environment, the authors compare their proposal with the performance obtained by op-

timizing either the accuracy or the communication resources, demonstrating that their framework leads to fairer solutions.

System heterogeneity Heterogeneous capacities of edge devices are a crucial problem. Some may be equipped with a GPU, while others only have the CPU. Some may be interconnected via unreliable wireless channels, whereas others through optical fiber backhaul links. System heterogeneity is the special focus of papers [27]–[30]. The authors of [27] first theoretically analyze the convergence rate of distributed gradient descent algorithms and then develop an adaptive control framework to find the best tradeoff between the number of local steps at the distributed nodes (the clients) and the number of model aggregation rounds, optimizing theoretical learning bound subjected to a local energy budget. The proposed network controller is evaluated on both convex problems and neural networks, obtaining close to optimal performance. Of particular note, this also holds for models based on Support Vector Machines (SVMs). The same objective is investigated in [28], where client selection is also addressed besides the number of local steps for the clients. The problem is formulated as multi-objective optimization, considering a weighted energy and completion time average, subject to an expected accuracy constraint. A real testbed implementation comprising 30 devices with heterogeneous computing capacities has been carried out, and results for neural networks training show that the proposed scheme is faster to converge and globally consumes less energy against competing solutions while also achieving a better model accuracy. The approach suggested in [29] additionally involves personalization of the gradient sparsification strategy for each client, where the (gradient) compression parameters are decided by evaluating the available energy resources for computation and communication. The proposed scheme drains significantly less energy with respect to benchmark algorithms in the presence of highly heterogeneous systems, almost without affecting the test error. An RL framework for client selection called AutoFL is proposed in [30] to obtain a data-driven solution to the problem of system heterogeneity with the objective of optimizing training time and energy efficiency. Taking as input the resources that are locally available to each client, AutoFL obtains a speed gain of 3.6 and an energy efficiency gain of about 5 times over traditional methods.

Fully decentralized learning The research described in the previous paragraphs considers federated learning solutions, where a central aggregator is in charge of updating the model from inputs received from a set of clients. The world of *fully decentralized learning* is instead widely unexplored when it comes to energy optimization. Although context-agnostic methods can always be applied to improve energy efficiency and the utilization rate of resources, an essential and unique feature of decentralized learning is the possibility of adjusting the communication network topology. In [31], the authors select a subset of the available links to perform model broadcasting so as to minimize the transmission power levels. A constraint on the minimum number of links required to guarantee the convergence of learning is added. This results in a combinatorial NP-hard problem over a graph that is relaxed to obtain an approximate solution. The considered scenarios involve the presence of interference and packet collisions. The results show that the proposed optimization can reduce energy consumption by more than 20% compared to simple heuristics,

without impacting the model performance.

1.3 On distributed optimization

This section is devoted to providing the reader with the minimum required mathematical background on distributed optimization to understand the main body of this thesis. The algorithms of the *dual ascent* and the Alternating Direction Method of Multipliers (ADMM) will be introduced and described in a general way here whereas they will be deepened and specialized for the specific optimization problems in chapters 2 and 3. The reader might refer to [32] for a more complete treatment of the topic.

1.3.1 The dual ascent

The dual ascent is a widely used method to solve constrained convex problems of the type

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & A_1\boldsymbol{x} = \boldsymbol{b}_1 \\ & A_2\boldsymbol{x} \leq \boldsymbol{b}_2, \end{aligned} \tag{1.3}$$

where $\boldsymbol{x} \in \mathbb{R}^d,\ d \geq 1$ is the optimization variable. Thus, with the dual ascent, one can easily handle both equality and inequality constraints. To solve this constrained problem we can resort to the Lagrangian multipliers method, which consists of the introduction of new variables λ and μ, namely, the Lagrange multipliers and solve the *dual* unconstrained problem

$$\sup_{\lambda,\mu_i \geq 0} \inf_{\boldsymbol{x}} \mathcal{L}\left(\boldsymbol{x}, \lambda, \mu\right), \tag{1.4}$$

where

$$\mathcal{L}\left(\boldsymbol{x}, \lambda, \mu\right) = f(\boldsymbol{x}) + \lambda^T\left(A_1\boldsymbol{x} - \boldsymbol{b}_1\right) + \mu^T\left(A_2\boldsymbol{x} - \boldsymbol{b}_2\right) \tag{1.5}$$

is the Lagrangian associated to problem (1.3). Under sufficiently mild conditions, it can be proven that problem (1.4) is equivalent to (1.3). The dual ascent algorithm solves this problem iteratively using gradient ascent with the following steps:

$$\boldsymbol{x}^+ = \operatorname*{argmin}_{\boldsymbol{x}} \mathcal{L}\left(\boldsymbol{x}; \lambda, \mu\right), \tag{1.6}$$

$$\lambda^+ = \lambda + \alpha\left(A_1\boldsymbol{x}^+ - \boldsymbol{b}_1\right), \tag{1.7}$$

$$\mu^+ = \max\left\{\lambda + \alpha\left(A_2\boldsymbol{x}^+ - \boldsymbol{b}_2\right), \boldsymbol{0}\right\}. \tag{1.8}$$

The $+$ operator stands for the update at the following algorithm iteration and the max in (1.8) is to be intended element-wise. It constrains the Lagrange multiplier μ_i to be null if the i-th inequality constraint is inactive at $\boldsymbol{x}^+$. The coefficient α is the step size and tunes the convergence speed

and precision of the solution. It must be sufficiently small to ensure the convergence (see [33] for more details on this).

This tool can be used as the *dual decomposition* in special cases where f is *separable*, i.e., $f(\boldsymbol{x}) = \sum_{m=1}^{M} f_m(\boldsymbol{x}_m)$, with $\boldsymbol{x}_m$ subvectors of $\boldsymbol{x}$. In this way, the primal step (1.6) becomes

$$x_m^+ = \underset{\boldsymbol{x}_m}{\operatorname{argmin}} \mathcal{L}\left(\boldsymbol{x}_m; \lambda, \mu\right). \tag{1.9}$$

This algorithm can be used by network agents to optimize their private local objective f_m when the global objective is f. The problem is coupled by the constraints and the agents, therefore, need to *broadcast* and *gather* the value of the Lagrange multipliers to update the price of the primal for the following iteration. The procedure can be hence solved via *message passing*.

1.3.2 The alternating direction method of multipliers

The ADMM was introduced to preserve the decomposability of the problem coming from the dual ascent and the better convergence characterization of the method of multipliers, another ancestor. Given two variables $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{z} \in \mathbb{R}^m$, we want to solve the problem

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{z}} \quad & f(\boldsymbol{x}) + g(\boldsymbol{z}) \\ \text{s.t.} \quad & A\boldsymbol{x} + B\boldsymbol{z} = \boldsymbol{c}. \end{aligned} \tag{1.10}$$

A fundamental assumption here is that the main original variable can be split in two and the objective can be written as a sum of independent functions of these two variables. The *augmented Lagrangian* associated to (1.10) is

$$\mathcal{L}\left(\boldsymbol{x}, \boldsymbol{z}, \lambda\right) = f(\boldsymbol{x}) + g(\boldsymbol{z}) + \lambda^T \left(A\boldsymbol{x} + B\boldsymbol{z} - \boldsymbol{c}\right) + \frac{\rho}{2} \left\|A\boldsymbol{x} + B\boldsymbol{z} - \boldsymbol{c}\right\|^2 \tag{1.11}$$

and the iterative steps to solve the problem are

$$\boldsymbol{x}^+ = \underset{\boldsymbol{x}}{\operatorname{argmin}} \mathcal{L}\left(\boldsymbol{x}; \boldsymbol{z}, \lambda\right), \tag{1.12}$$

$$\boldsymbol{z}^+ = \underset{\boldsymbol{z}}{\operatorname{argmin}} \mathcal{L}\left(\boldsymbol{z}; \boldsymbol{x}^+, \lambda\right), \tag{1.13}$$

$$\lambda^+ = \lambda + \rho\left(A\boldsymbol{x}^+ + B\boldsymbol{z}^+ - \boldsymbol{c}\right). \tag{1.14}$$

Hence, this algorithm consists in splitting the objective and separating the optimization in $\boldsymbol{x}$ and $\boldsymbol{z}$, from which the name "alternating direction". Again this method can be used in networks to solve distributedly optimization problems via message passing. An advanced version of this, namely, the Douglas-Rachford splitting, is detailed and specialized for the specific network problem of Chapter 2 in this thesis.

1.4 Thesis contributions

This thesis advances the state-of-the-art of the design, management, and operation of sustainable MEC networks with the original contributions detailed in the following list.

- The abstract design of an edge agent comprising i) computing equipment to provide the users with processing power, ii) a wireless communication unit to exchange data with the users, iii) connection to wired backhaul links to offload tasks to other edge agents, and iv) connection to the power grid and to EH devices, possibly having also the availability of an energy accumulator (battery).
- The proposal of a fully decentralized online and predictive model-based optimization framework to i) plan the local execution of the jobs, ii) decide the portions to offload to neighboring servers, and iii) trade electrical energy with the power grid. This is done with the general objective of minimizing the carbon footprint to manage in the best possible ways the RERs (Chapter 2).
- The design of solutions for load balancing and server consolidation (i.e., try to switch some servers off to save further energy) (Chapter 2).
- The introduction into the system of users' mobility in an urban vehicular scenario, increasing the carbon footprint minimization objective with the requirement of following the mobility patterns when possible (Chapter 3).
- A comparison study of the communication and computation overhead between solving entirely distributedly the problem and performing first a local estimation of resources and solving the global problem on average afterwards – which is the difference between the solutions proposed in Chapters 2 and 3, respectively (Chapter 4).

More specific contributions for each chapter will be detailed therein.

1.5 Thesis organization

The main body of this thesis is organized into three chapters, presenting my main two scientific contributions, published in international journals. Chapter 2 deals with the problem of jobs execution and offloading in a hierarchical MEC platform with servers equipped with a battery and PV panels and further connected to the power grid [9]. Chapter 3 presents Energy-Aware job Scheduling at the Edge (EASE), an algorithm developed for the jobs execution and migration in urban MEC with user mobility [19]. Here the energy objective is increased with the additional goal of following the vehicles' trajectories during the handovers. Chapter 4 is a comparison of the optimization and control methods used in the two works in terms of network overhead and computational efficiency. The thesis ends by drawing some concluding remarks and outlining future research directions with Chapter 5.

2

Managing renewable energy resources at the network edge

2.1 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. The 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) [34]. 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 MCC model towards a highly distributed MEC one, where computing power, network control, and storage are pushed to the network edge [35]. What is often ignored, is that this computation drains a noticeable amount of energy, drastically increasing the carbon footprint of mobile networks [36].

MEC is inherently more energy-efficient than its cloud computing counterpart, as computing units are moved closer to the User Equipment (UE), and, in turn, the consumption associated with communicating the data to the servers is drastically reduced. Moreover, edge servers do not need powerful cooling systems, which cause about 40% of the total energy consumption of large data centers [37], and represent a root cause of global overheating. Despite this, MEC alone is unable to solve the excessive energy consumption caused by computing tasks: sizeable benefits can only be obtained through careful management of the available energy and computing resources, by shifting the workload over time to the more energy-efficient computing units and, at the same time,

taking advantage of the recent developments and portability of energy harvesting devices. The objective is to distributedly control hybrid systems with harvesting and computing capabilities, promoting the transition towards energy-neutral edge networks.

Hence, in the present chapter, 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 [35], 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. The survey in [38] deals with solar-powered BSs and their open issues and shows the worldwide distribution of such BSs as of 2014. The industry has paid attention to this topic already for some years: LG built a 4G LTE network with self-sustained BSs in 2016 [39], and Moxa proposes market-ready hybrid solar-grid solutions for IoT in smart-homes [40]. The Academia also recently started to build real testbeds of hybrid-powered edge systems [41], [42].

In this chapter, 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*) [43] 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 ADMM [32].

The main contributions of the present chapter are:

- 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.
- 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 re-distribution (load balancing).
- 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 based on *Model Predictive Control* (*MPC*), which uses future job and energy arrival estimates, obtained via low-complexity predictors.
- For scalability purposes, a fully distributed iterative procedure for solving the predictive control problem is proposed, based on the DRS algorithm.

- 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 chapter is organized as follows. Section 2.2 discusses the related works. Section 2.3 presents the system model. The optimization problem and its distributed solution are detailed in Sections 2.4 and 2.5, respectively. Numerical results are presented in Section 2.6, and concluding remarks are given in Section 2.7.

2.2 State of the art

The multi-access edge computing paradigm has received considerable attention from academia [35], [36], [44]–[46] and industry [47] alike. The typical objectives pursued by MEC resource allocation algorithms are: minimizing power consumption, minimizing the execution delay, or maximizing revenue. In some studies, these objectives are combined, e.g., through a weighted sum of power consumption and delay [48]. In this chapter, we propose a distributed, online, and adaptive optimization framework for computation load offloading in a network of edge servers, to manage 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., [15], [49].

Server power consumption models and how they relate to CPU load are the focus of [50]–[52]. 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 to design these systems to be as *energy-efficient* as possible [15], [49], [53]. In [49], 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 MPC based framework. The same technique with a similar system model is used also in [15], where the authors build a dynamic, adaptive, and online resources management scheme, but the approach here is centralized. The authors of [53] propose a Suc-

cessive 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 correspond to globally minimizing the energy used by the system but to optimally exploiting the energy coming from renewable sources at each server. Note that the two objectives do not necessarily coincide: with renewables it is at times optimal to drain as much energy as possible, to minimize the energy that goes unused, and thus lost. According to [50]–[52], 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 EH technologies [36], [48], [54]–[57]. The authors of [48] 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 chapter, their framework requires a centralized controller, which is a strict requirement in edge networks. The authors of [54] and [57] 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 [36], 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 [54] and [57], 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 [58].

MPC-based approaches have been previously used for controlling networked systems [58]–[62]. Data centers VM management is the focus of [59], which presents an energy-aware consolidation strategy. A resource management approach effectively capturing the non-linear behavior in VM resource usage through fuzzy modeling is presented in [60]. 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 [58] 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 propose a strategy to solve the problem in a distributed

fashion.

In conclusion, the approaches found in the literature present the following main drawbacks: i) most commonly, the proposed methods are centralized, or require an omniscient orchestrator; ii) they do not adapt to varying workload intensities and arrival statistics; iii) often, they consider load-balancing as the sole optimization metric, ignoring other objectives (such as server consolidation); iv) minimization is carried out targeting the global energy consumption, which may be suboptimal for the management of renewables in hybrid MEC systems, as it may be convenient to drain more energy and maximize the consumption at those computing units with abundant energy inflow. In contrast, we propose an online and predictive optimization framework based on MPC, whose goal is to reduce the electric energy drainage from the power grid, while using the harvested energy as much as and where possible. To this end, we present a fast and customized version of the ADMM algorithm, which makes the solution completely distributed, using message-passing among one-hop neighbor servers.

2.3 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 BSs that receive job processing requests from end-users. An access node $i \in \mathcal{M}_a$ receiving a computing request can either process *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 chapter, 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, which 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 chapter is summarized in Table 2.1, whereas the main blocks of the server architecture are shown in Fig. 3.2.

Table 2.1: List of symbols used in the chapter.

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
$\overline{s}_i$	maximum amount of buffered workload at server i
$e_{i,k}$	amount of energy stored at server i (energy buffer state)
$\overline{b}_i$	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
$\overline{w}_i$	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
$t^d_{i,k}$	job (CPU cycles) with deadline d at server i, at time k
δ^{E}_i	discount factor accounting for energy decay
F	Jain's fairness index, defined in (2.37)
ε	Job generation rate
$\varphi_{i,k}$	load factor at server i at time k, defined in (2.15)
η	efficiency metric, defined in (2.38)
$\mathcal{P}_\ell$	proximity operators, $\ell \in \{1, 2\}$
$\mathcal{Q}_\ell$	reflected proximity operators, $\ell \in \{1, 2\}$
T_m	edge network tier, $m \in \{0, 1, 2\}$
$\zeta^{\mathrm{OFF}}_{i,j}$, ζ^{CPU}_i	workload to energy consumption conversion factors
$\zeta^{\mathrm{PUR}}_{i,k}$, $\zeta^{\mathrm{SOLD}}_{i,k}$	cost of purchasing (PUR) and selling (SOLD) energy

2.3.1 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 $t^d_{i,k}$. Note that $t^d_{i,k}$ represents the number of CPU cycles that are required to complete these tasks, referred to in the following as the job *intensity* at time k. For any (i,k,d), the intensity of a new job arrival is picked from a generic probability mass function, see Section 2.3.4. We also consider a maximum deadline of D slots, and, accordingly, we denote by $\boldsymbol{t}_{i,k} = [t^1_{i,k}, \ldots, t^d_{i,k}, \ldots, t^D_{i,k}]^T$ the column vector containing the generated workload for all possible execution deadlines $d \in \mathcal{D} = \{1, 2, \ldots, D\}$ at node i and time k.

According to our model, each new job has an associated execution deadline d, which means that, from the time slot in which the job arrives, the system has at most d slots available to execute it. So, if a job with deadline d arrives at time slot k, it will have to be executed anytime before slot $k+d$. A time-sensitive service is one that provides some delay guarantees, either in a probabilistic or deterministic sense. In this work, we adopt a deterministic approach, by requiring the strict execution of the job before the associated deadline expires. 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. Note that the external process $\boldsymbol{t}_i$ is instantaneous, i.e., it collects new arrivals for the current slot only. Nonetheless, once jobs enter the system, they modify the state equation according to Eq. 2.1, using resources until either the task has been processed or has expired. We remark that jobs are heterogeneous as their intensity is allowed to vary for each node, at each time slot, and for each deadline. Further details on the stochastic model governing the arrivals are provided in Section 2.3.4.

2.3.2 Edge server architecture

Each edge server i (see Fig. 3.2) has a maximum storage capacity $\overline{s}_i$, representing the maximum amount of buffered workload at the node, and a processing rate $\overline{w}_i$, representing the number of CPU cycles per time slot that can be executed (Table 2.1). These quantities are free to vary across servers.

The computational power of server i is shared by:

i. newly generated jobs that 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 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

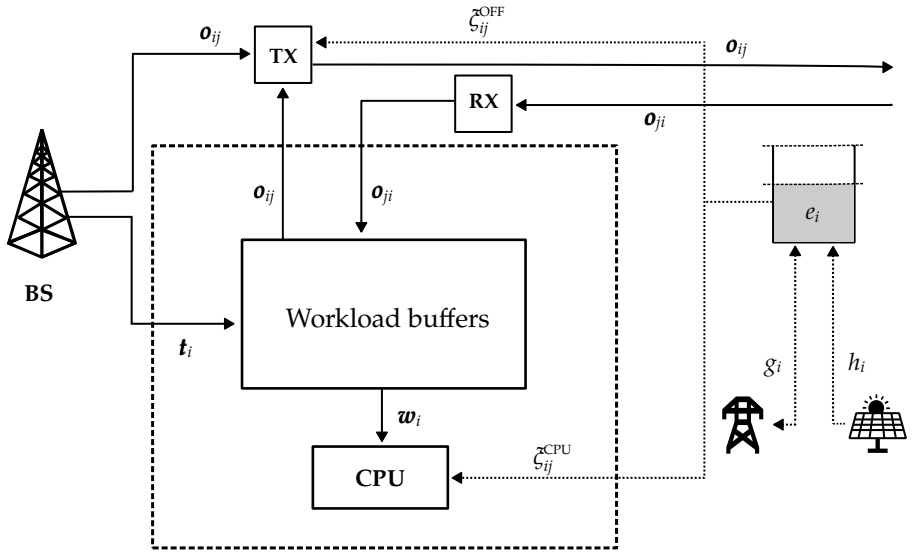


Figure 2.1: Edge server architecture. Edge servers are co-located with the access points at BSs, and directly receive jobs from end devices. The servers can process the incoming workload locally or offload it to other servers. They are also equipped with a battery, which is charged by the energy harvested from the environment (h_i). Besides, the energy unit is also connected to the power grid, allowing the purchase and sale of energy (g_i).

buffers, where each buffer is devoted to meeting a specific deadline d, as depicted in Fig. 2.2. 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 in time slot k. We also assume that the expiring backlogged workload, with a deadline of $d = 1$, cannot be offloaded.

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

$$s_{i,k+1}^{d-1} = s_{i,k}^d + \underbrace{t_{i,k}^d}_{\text{locally generated}} - \overbrace{w_{i,k}^d}^{\text{locally executed}} + \underbrace{\sum_{j \in \mathcal{N}_i} o_{ji,k}^d}_{\text{offloaded here}} - \overbrace{\sum_{j \in \mathcal{N}_i} o_{ij,k}^d}^{\text{offloaded elsewhere}}, \tag{2.1}$$

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 (2.1) we explicitly account for the tasks locally generated at the access node, $t_{i,k}^d$ where

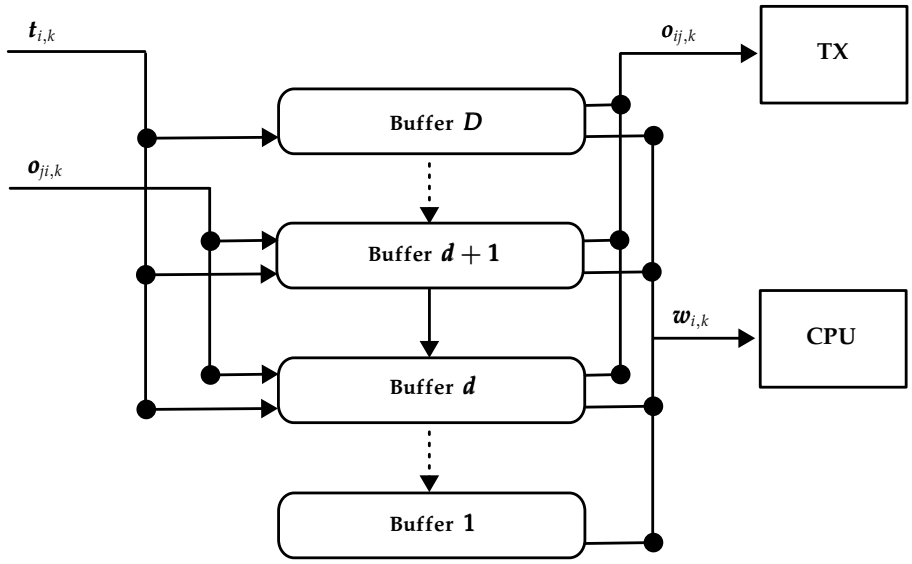


Figure 2.2: Detail of workload buffers. At each server i, workload can be received from a neighbor ($\boldsymbol{o}_{ji,k}$) or generated locally ($\boldsymbol{t}_{i,k}$). The incoming workload is sent to the appropriate buffer, according to its execution deadline. Workload $\boldsymbol{w}_{i,k}$ is executed locally, whereas $\boldsymbol{o}_{ij,k}$ is offloaded to a neighbor server.

$t^d_{i,k} = 0$ if $i \notin \mathcal{M}_a$, the tasks that are locally executed, $w^d_{i,k}$, those incoming from neighboring nodes, $o^d_{ji,k}$, and the amount of workload that is outsourced $o^d_{ij,k}$.

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

$$\boldsymbol{T}_D = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & 0 & 1 \\ & & & 0 \end{bmatrix} \tag{2.2}$$

of size $D \times D$, having ones in the sup-diagonal and zeros elsewhere, (2.1) can be compactly rewritten in vector form as

$$\boldsymbol{s}_{i,k+1} = \boldsymbol{T}_D \left(\boldsymbol{s}_{i,k} + \boldsymbol{t}_{i,k} - \boldsymbol{w}_{i,k} + \sum_{j \in \mathcal{N}_i} \left(\boldsymbol{o}_{ji,k} - \boldsymbol{o}_{ij,k} \right) \right), \tag{2.3}$$

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. 2.2 for a pictorial representation of the relations between these variables.

2.3.3 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 in a local energy buffer. The use of renewable energy from an energy source different than the co-located one is not possible. 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 $e_{i,k}$, $k = 0, 1, \ldots$, at node $i \in \mathcal{M}$ evolves according to

$$e_{i,k+1} = \delta_i^{\mathrm{E}} e_{i,k} + h_{i,k} + g_{i,k} - \sum_{j \in \mathcal{N}_i, d \in \mathcal{D}} \xi_{ij}^{\mathrm{OFF}} o_{ij,k}^d - \sum_{d \in \mathcal{D}} \xi_i^{\mathrm{CPU}} w_{i,k}^d \tag{2.4}$$

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.

2.3.4 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). Arrival 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 $k+1, k+2, \ldots$ 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 $\hat{f}_i^{\text{MC}}$ and with intensity $\overline{p}_i^{\text{MC}}$, and no arrivals with probability $1 - \hat{f}_i^{\text{MC}}$.

2.4 Problem formulation

2.4.1 Model predictive control framework

The system state equations (2.3) and (2.4) constitute the backbone for an MPC framework [43] 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 immediately at the following step $k+1$ and the procedure is repeated, 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.

2.4.2 Workload buffers evolution

We preliminarily generalize (2.3) in a form that projects the buffers' state forward in time by n time slots. By iterated application of (2.3), we obtain

$$\boldsymbol{s}_{i,k+n} = \boldsymbol{T}_D^n \boldsymbol{s}_{i,k} + \sum_{m=0}^{n-1} \boldsymbol{T}_D^{n-m} \left(\boldsymbol{t}_{i,k+m} - \boldsymbol{w}_{i,k+m} + \sum_{j \in \mathcal{N}_i} (\boldsymbol{o}_{ji,k+m} - \boldsymbol{o}_{ij,k+m}) \right), \tag{2.5}$$

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

We compactly write (2.5) 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 (2.5) over the prediction horizon, and for the whole network, is expressed by the linear relation

$$\boldsymbol{s} = \boldsymbol{A}_w \boldsymbol{s}_1 + \boldsymbol{B}_w (\boldsymbol{t} - \boldsymbol{w} + \boldsymbol{C}'_w \boldsymbol{o} - \boldsymbol{C}''_w \boldsymbol{o}) \,, \tag{2.6}$$

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 (2.5). Specifically, $\boldsymbol{C}'_w$ and $\boldsymbol{C}''_w$ are binary matrices

that respectively collect the sums for $j \in \mathcal{N}_i$ in (2.5). Instead, for matrices $\boldsymbol{A}_w$ and $\boldsymbol{B}_w$, we have

$$\boldsymbol{A}_w = \boldsymbol{I}_M \otimes \left(\sum_{k=1}^{N} \boldsymbol{e}_{k,N} \otimes \boldsymbol{T}_D^k \right), \tag{2.7}$$

$$\boldsymbol{B}_w = \boldsymbol{I}_M \otimes \left(\sum_{k=1}^{N} \boldsymbol{T}_N^{k-1} \otimes \boldsymbol{T}_D^k \right) \tag{2.8}$$

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 (2.6) 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 the 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.

Some side constraints that govern the behavior of (2.6) further need to be formalized.

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

$$w_{i,k}^d, o_{ij,k}^d \geq 0 \tag{2.9}$$

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 (due to the physical transmission constraints). We therefore assume

$$\sum_{d \in \mathcal{D}/\{1\}} o_{ij,k}^d \leq \bar{o}_{ij} \tag{2.10}$$

for each node $i \in \mathcal{M}$, neighbor node $j \in \mathcal{N}_i$, and slot k. We assume, for analytical tractability, that the amount of transferred data is proportional to the workload exchanged o. With this assumption, this bound acts as a rate constraint.

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

$$0 \leq s_{i,k}^d \leq \bar{s}_i \,. \tag{2.11}$$

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

$$w_{i,k}^d + \sum_{j \in \mathcal{N}_i} o_{ij,k}^d \leq s_{i,k}^d + t_{i,k}^d \tag{2.12}$$

applies, which ensures that nodes can process or offload only the existing workload, i.e, the backlogged ($s_{i,k}^d$) or the newly generated ($t_{i,k}^d$) one. This corresponds to the assumption that the offloaded workload $o_{ij,k}^d$ takes a full-time slot to be delivered to node j, and it will therefore be avail-

able for execution (in buffer $d-1$) starting from time $k+1$. Thus, workload offloading delays the 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

$$w_{i,k}^1 = s_{i,k}^1 + t_{i,k}^1\,, \quad o_{ij,k}^1 = 0 \tag{2.13}$$

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 $\overline{w}_i$ CPU cycles per time slot, then it must hold

$$\sum_{d\in\mathcal{D}} w_{i,k}^d \leq \overline{w}_i \tag{2.14}$$

for every edge server $i \in \mathcal{M}$ and time slot k. We remark that (2.14) and (2.12), together with (2.9), upper bound the amount of workload that can be moved across nodes, o, and locally executed, w.

Based on the processing capacity, the *load factor* of server i in slot k is

$$\varphi_{i,k} = \frac{\sum_{d\in\mathcal{D}} w_{i,k}^d}{\overline{w}_i}\,. \tag{2.15}$$

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

2.4.3 Energy buffers evolution

The evolution of the energy buffers (2.4) 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,

$$\boldsymbol{e} = \boldsymbol{A}_e\boldsymbol{e}_1 + \boldsymbol{B}_e(\boldsymbol{h} + \boldsymbol{g} - \boldsymbol{C}_e'\boldsymbol{o} - \boldsymbol{C}_e''\boldsymbol{w}) \tag{2.16}$$

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 (2.16) assume a form similar to that of (2.7), namely

$$\boldsymbol{A}_e = \boldsymbol{I}_M \otimes \left(\sum_{k=1}^{N} (\delta_i^{\mathrm{E}})^k\, \boldsymbol{e}_{k,N}\right) \tag{2.17}$$

$$\boldsymbol{B}_e = \boldsymbol{I}_M \otimes \left(\sum_{k=1}^{N} (\delta_i^{\mathrm{E}})^k\, \boldsymbol{T}_N^{k-1}\right) \tag{2.18}$$

while $\boldsymbol{C}'_e$ and $\boldsymbol{C}''_e$ are binary matrices collecting the two sums of (2.4). 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 $\overline{b}_i$, and a minimum working threshold $\underline{b}_i$, that is

$$\underline{b}_i \leq e_{i,k+1} \leq \overline{b}_i \tag{2.19}$$

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

2.4.4 Objective functions

To facilitate the reader in following the flow of the Chapter, we recall here the meaning of the optimization variables used in the formalization of the optimization problem. i) $\boldsymbol{g}$ is the amount of electrical energy traded with the power grid; ii) $\boldsymbol{o}$ is the quantity of workload offloaded to neighboring servers; iii) $\boldsymbol{w}$ is the quantity of workload executed locally. As aforementioned, MPC solves at every time step an optimization problem for every node $i \in \mathcal{M}$, minimizing some predefined cost which is a 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, (2.6) will be denoted more explicitly in the form $\boldsymbol{s}(\boldsymbol{o}, \boldsymbol{w})$, and (2.16) in the form $\boldsymbol{e}(\boldsymbol{g}, \boldsymbol{o}, \boldsymbol{w})$. This leads to the *centralized* optimization problem

$$\begin{aligned} P_1: \quad & \min_{\boldsymbol{g},\boldsymbol{o},\boldsymbol{w}} J(\boldsymbol{g}, \boldsymbol{o}, \boldsymbol{w}) \\ & \text{s.t. } (2.9) - (2.14), \text{ and } (2.19) \text{ hold,} \end{aligned} \tag{2.20}$$

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

In this chapter, two processing cost functions are compared: a quadratic and a logarithmic one, the latter leading to a non-convex optimization problem. These have different purposes and should be seen as complementary. Whether to choose the quadratic or the logarithmic one is up to the service provider, as they force the system to behave in opposite manners. The quadratic function promotes load balancing, whereas the logarithmic one promotes consolidation, i.e., reduces, as much as possible, the number of active servers.

For both functions, the importance of the workload buffers cost minimization concerning the operation cost is tuned through the 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

$$J(\boldsymbol{g}, \boldsymbol{o}, \boldsymbol{w}) = (1-\gamma)\|\boldsymbol{s}(\boldsymbol{o}, \boldsymbol{w})\|^2_{\boldsymbol{Q}_s} + \gamma\Big(\mathcal{R}_e(\boldsymbol{g}) + \|\boldsymbol{o}\|^2_{\boldsymbol{R}_o} + \langle \boldsymbol{r}_o, \boldsymbol{o}\rangle + \|\boldsymbol{w}\|^2_{\boldsymbol{R}_w}\Big), \tag{2.21}$$

where $\|\boldsymbol{x}\|^2_{\boldsymbol{Q}} = \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$ 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. To better explain, let us consider a fully-connected mesh network, and let $\boldsymbol{o}$ contain the amount of workload

o_{ij}^d that is transmitted from server i to server j, expressed in vector form as

$$\begin{aligned}\boldsymbol{o} = [&o_{11}^1, \dots, o_{1M}^1, o_{21}^1, \dots, o_{2M}^1, \dots, o_{M1}^1, \dots, o_{MM}^1,\\ &, o_{11}^2, \dots, o_{1M}^2, o_{21}^2, \dots, o_{2M}^2, \dots, o_{M1}^2, \dots, o_{MM}^2, \dots,\\ &, o_{11}^D, \dots, o_{1M}^D, o_{21}^D, \dots, o_{2M}^D, \dots, o_{M1}^D, \dots, o_{MM}^D]^T,\end{aligned}$$

where M is the number of servers. If matrix $\boldsymbol{R}_o$ is diagonal, then its element $(\boldsymbol{R}_o)_{\ell\ell} \geq 0$, $\ell \in \{1, \dots, M^2 D\}$, represents the cost of transmitting one unit of flow o_{ij}^d from buffer (deadline) $d = \lfloor (\ell - 1)/M^2 \rfloor + 1$ from server i to server j, where

$$\begin{aligned} i &= \lfloor (\ell - (d-1)M^2 - 1)/M \rfloor + 1 \\ j &= [(\ell - (d-1)M^2 - 1) \mod M] + 1. \end{aligned}$$

Non-diagonal cost matrices allow defining correlated cost components for the same sources ℓ or destinations m. The same approach and considerations hold for matrix $\boldsymbol{R}_w$.

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

$$\mathcal{R}_e(\boldsymbol{g}) = \sum_{i \in \mathcal{M}, k \in \mathcal{T}} \mathcal{R}_{i,k}(g_{i,k}) \ , \quad \mathcal{R}_{i,k}(x) = \begin{cases} \xi_{i,k}^{\text{PUR}}\, x, & x \geq 0 \\ -\xi_{i,k}^{\text{SOLD}}\, x, & x < 0 \end{cases} \tag{2.22}$$

where we assume $0 < \xi_{i,k}^{\text{SOLD}} < \xi_{i,k}^{\text{PUR}}$, 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 (2.21), P_1 can be solved in a centralized manner using a constrained Quadratic Programming (QP) solver. As a matter of fact, the constraints in (2.20) are all linear, all the weighted norms in (2.21) have a quadratic expression thanks to the fact that (2.6) and (2.16) are linear, and the energy cost in (2.22) 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 the proximity of the zero, a sleeping server will prefer to offload the workload to an already working neighbor if it can avoid turning its processing unit on. With the same notation employed for the quadratic cost, the logarithmic objective function is defined as

$$J(\boldsymbol{g}, \boldsymbol{o}, \boldsymbol{w}) = (1-\gamma)\langle \boldsymbol{q}_s, \boldsymbol{s}(\boldsymbol{o}, \boldsymbol{w}) \rangle + \gamma \Big(\mathcal{R}_e(\boldsymbol{g}) + \langle \boldsymbol{r}_o, \boldsymbol{o} \rangle + \sum_{i \in \mathcal{M}} \log(1 + \langle \boldsymbol{r}_{w,i}, \boldsymbol{w}_i \rangle) \Big), \tag{2.23}$$

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. These vectors are the counterparts of matrices $\boldsymbol{R}_o$ and $\boldsymbol{R}_w$ defined above: they allow customizing the transmission cost for each link and the processing cost for each server. Note that the cost function in (2.23) is non-convex and, in turn, heuristic methods must be used to solve the associated optimization problem.

2.5 Decentralized problem solution

2.5.1 Background on Distributed Optimization

In this thesis, *message passing* techniques are used, 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 [32] has recently received 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 [63]–[65]. 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 [63], [64]. 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 [65], [66]. Additionally, ADMM-like strategies can be heuristically used to deal with non-convex problems [32], [67].

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 [66], [68], [69]. In [68], different algorithms are compared in terms of convergence speed, number of messages exchanged, and distributed computation architecture. The authors of [66] propose an ADMM-based algorithm to solve the D-MPC problem, and the effects of prematurely terminating the iterative ADMM procedure are investigated in [69].

In this chapter, a customized DRS of the centralized MPC problem defined in (2.20) is derived, optimizing a convex and a non-convex cost function in a distributed fashion over a given topology.

2.5.2 ADMM framework for scheduling computing jobs

The centralized problem P_1 in (2.20) 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

$$\boldsymbol{x} = [\boldsymbol{g}; \boldsymbol{o}; \tilde{\boldsymbol{o}}; \boldsymbol{w}] \tag{2.24}$$

and the associated sub-space

$$\mathcal{X} = \{\boldsymbol{x} | \tilde{\boldsymbol{o}} = \boldsymbol{o}\} \ , \tag{2.25}$$

which identifies an added linear constraint to be solved. Accordingly, the cost functions and the linear constraints in (2.20) must be slightly modified to obtain the said separation. This simply requires replacing $o^d_{ij,k}$ with $\tilde{o}^d_{ij,k}$ throughout the expressions of Section 3.5, 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 (2.6) assumes the alternative form

$$\boldsymbol{s}(\boldsymbol{x}) = \boldsymbol{A}_w \boldsymbol{s}_1 + \boldsymbol{B}_w(\boldsymbol{t} - \boldsymbol{w} + \boldsymbol{C}'_w \tilde{\boldsymbol{o}} - \boldsymbol{C}''_w \boldsymbol{o}) \ , \tag{2.26}$$

while the energy buffer update (2.16) is not modified (since it does not use $\tilde{\boldsymbol{o}}$). Also, the constraints (2.9)-(2.14) and (2.19), as well as the cost expressions (2.21) and (2.23), are not modified but for the fact that state variables now exploit (2.26). Hence, problem P_1 can be rewritten in the equivalent form

$$\begin{aligned} P_2 : \quad & \min_{\boldsymbol{x}} J(\boldsymbol{x}) \\ & \text{s.t. } \boldsymbol{x} \in \mathcal{X} \\ & \quad \boldsymbol{x} \in \mathcal{Y} = \{\boldsymbol{x} | (2.9) - (2.14) \text{ and } (2.19) \text{ hold}\} \end{aligned} \tag{2.27}$$

where the linear constraints of problem (2.20) are collected in the (linear) sub-space $\mathcal{Y}$.

2.5.3 ADMM algorithm

P_2 in (2.27) 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 parallelized 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

$$\begin{aligned} P_3 : \quad & \min_{\boldsymbol{x}} \underbrace{J(\boldsymbol{x}_1) + \mathcal{I}_{\mathcal{Y}}(\boldsymbol{x}_1)}_{f_1(\boldsymbol{x}_1)} + \underbrace{\mathcal{I}_{\mathcal{X}}(\boldsymbol{x}_2)}_{f_2(\boldsymbol{x}_2)} \\ & \text{s.t. } \boldsymbol{x}_1 = \boldsymbol{x}_2 \end{aligned} \tag{2.28}$$

where

$$\mathcal{I}_{\mathcal{X}}(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} \in \mathcal{X} \ , \\ +\infty & \text{otherwise} \, . \end{cases} \tag{2.29}$$

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

The solution to (2.28) 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 [63], [64]), illustrated in Algorithm 2.1, where the $^+$ sign denotes an updated variable, ρ and q are tuneable parameters that control the convergence speed, and

$$\mathcal{P}_\ell(\boldsymbol{u}) = \underset{\boldsymbol{x}}{\operatorname{argmin}}\ f_\ell(\boldsymbol{x}) + \frac{\rho}{2}||\boldsymbol{x}-\boldsymbol{u}||^2\,, \tag{2.30}$$

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

Algorithm 2.1 Scaled variables ADMM

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

Nicely, the proximity operator (2.30) with $\ell = 2$ assumes the simple form

$$\mathcal{P}_2(\boldsymbol{u}) = \underset{\boldsymbol{x}\in\mathcal{X}}{\operatorname{argmin}}\ \frac{\rho}{2}||\boldsymbol{x}-\boldsymbol{u}||^2 = \boldsymbol{L}_{\mathcal{X}}\cdot\boldsymbol{u} \tag{2.31}$$

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

$$\boldsymbol{L}_{\mathcal{X}} = \begin{bmatrix} \boldsymbol{I}_G & & & \\ & \frac{1}{2}\boldsymbol{I}_O & \frac{1}{2}\boldsymbol{I}_O & \\ & \frac{1}{2}\boldsymbol{I}_O & \frac{1}{2}\boldsymbol{I}_O & \\ & & & \boldsymbol{I}_W \end{bmatrix}\,, \tag{2.32}$$

where G, O, and W are the dimensions (lengths) of, respectively, $\boldsymbol{g}$, $\boldsymbol{o}$, and $\boldsymbol{w}$. Note that (2.32) is a projection operator that allows extracting an average value from $\boldsymbol{o}$ and $\tilde{\boldsymbol{o}}$.

Instead, the proximity operator (2.30) with $\ell = 1$ takes the form

$$\mathcal{P}_1(\boldsymbol{u}) = \underset{\boldsymbol{x}\in\mathcal{Y}}{\operatorname{argmin}}\ J(\boldsymbol{x}) + \frac{\rho}{2}||\boldsymbol{x}-\boldsymbol{u}||^2 \tag{2.33}$$

which, thanks to choice (2.24), 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

$$\mathcal{P}_{1,i}(\boldsymbol{u}) = \underset{\boldsymbol{x}_i\in\mathcal{Y}_i}{\operatorname{argmin}}\ J_i(\boldsymbol{x}_i) + \frac{\rho}{2}||\boldsymbol{x}_i-\boldsymbol{u}_i||^2\,, \tag{2.34}$$

where $\mathcal{Y}_i$ collects the constraints (2.9)-(2.14), and (2.19) for the ith node, and $J_i(\cdot)$ is the cost contribution ((2.21) or (2.23)) of node i.

2.5.4 Douglas-Rachford splitting version

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

$$\mathcal{Q}_\ell(\boldsymbol{u}) = 2\sqrt{\rho}\mathcal{P}_\ell(\boldsymbol{u}/\sqrt{\rho}) - \boldsymbol{u}\ , \tag{2.35}$$

Algorithm 2.1 can be equivalently rewritten in the extremely compact form of Algorithm 2.2, namely a DRS counterpart that tracks the unique variable $\boldsymbol{z} = \sqrt{\varepsilon}(\boldsymbol{y}^+ + \lambda)$ (see details of this formalization in [64]). The explicit relation with the system variables is in this case

$$\boldsymbol{x}_1^+ = \mathcal{P}_1(\mathcal{Q}_2(\boldsymbol{z})/\sqrt{\rho})\ , \quad \boldsymbol{x}_2\ = \mathcal{P}_2(\boldsymbol{z}/\sqrt{\rho})\ . \tag{2.36}$$

Algorithm 2.2 Douglas-Rachford Splitting counterpart

1: $\boldsymbol{z}^+\ = (1-q)\boldsymbol{z} + q\mathcal{Q}_1(\mathcal{Q}_2(\boldsymbol{z}))$

In Algorithm 2.2, $q \in [0,1]$, $\boldsymbol{z}$ is the global version of the variables $\boldsymbol{z}_i = \{\gamma_{i,k}, \{\sigma_{ij,k}, \tilde{\sigma}_{ji,k}\}_{j\in\mathcal{N}_i}, \omega_{i,k}\}$, where $\gamma_{i,k}$, $\sigma_{ij,k}$, $\tilde{\sigma}_{ji,k}$ and $\omega_{i,k}$ 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 (2.36). Algorithm 2.2 works on a unique state variable $\boldsymbol{z}$, as opposed to the three state variables of Algorithm 2.1, namely, $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, and λ, hence it is to be preferred from a computational perspective, also because the computational complexities of operators $\mathcal{P}_\ell$ and $\mathcal{Q}_\ell$ is identical. In addition, we empirically verified that, despite its simplicity (or, possibly, because of it), Algorithm 2.2 shows 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 [70] 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 the 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: $\left|\frac{f(\boldsymbol{x}_1)}{f(\boldsymbol{x}_1^+)}\right| - 1 < f_{\text{obj}}$.

2.5.5 Convergence properties

Quadratic cost function: when the quadratic objective function of (2.21) is used, (2.33) amounts to solving a convex quadratic problem with linear constraints. Any QP solver can be used to obtain local updates and the algorithm is ensured to converge linearly (e.g., see [63]) 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 [64].

Logarithmic cost function: when, instead, the non-convex logarithmic cost function (2.23) is used, then the target function in (2.33) 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 [67], which uses DC programming with linear approximation. Specifically, convergence is ensured under weak assumptions if the cost function in (2.33) 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.

2.5.6 Final distributed algorithm

Algorithm 2.2 is expressed in compact form thanks to the reflected proximity operators defined in (2.35). The final algorithm is written from a server perspective in Algorithm 2.3. Lines 7-8 of Algorithm 2.3 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 $\mathcal{Q}_2(\sigma_{ij}) = 2\mathcal{P}_2(\sigma_{ij}) - \sigma_{ij} = \tilde{\sigma}_{ij}$, with $\mathcal{P}_2(\sigma_{ji}) = (\sigma_{ij} + \tilde{\sigma}_{ij})/2$, and, analogously, $\mathcal{Q}_2(\tilde{\sigma}_{ji}) = \sigma_{ji}$. This means that the projected offloading variables of node i, σ_{ij} (from $i \to j$), must take the value of $\tilde{\sigma}_{ij}$ (indicating, at neighbor j, what is taken from i). Lines 10-11 correspond to using $\mathcal{Q}_1$, as well as extracting $\boldsymbol{x}_1^+$ according to (2.36). On line 14, the Anderson acceleration (A2DR in [70]) is applied, considering all the values of $\boldsymbol{z}_i$ from a circular buffer Z, of fixed length.

2.6 Numerical results

2.6.1 Simulation scenario and parameters

Scenario

As a reference scenario, we consider the three-tier network of Fig. 2.3, 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 2.3.4. For job arrivals, we consider bursts of length $b = \max(3, \varepsilon/(1-\varepsilon))$, with ε being the generation rate. Energy arrives in bursts of a fixed average length of 50 slots, following a correlated generation process with $\varepsilon_e = 0.6$. Data that needs processing is sent to the closest access server in tier T_1, where the actual optimization takes place. The results that follow, 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 a 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, the workload can be only sent from T_1 to T_2 servers, but it cannot be sent back

Algorithm 2.3 MPC based allocation of processing tasks

Input: convergence parameters ρ and q, stopping threshold f_{obj} and z_{obj}, cost function J (or $\tilde{J}$), buffer Z of fixed length
Output: workload allocation for the current step
initialize: $\boldsymbol{z} = \boldsymbol{0}$, $\boldsymbol{x} = \boldsymbol{0}$, $f_{\text{cur}} = \infty$, $z_{\text{cur}} = \infty$
while $f_{\text{cur}} > f_{\text{obj}}$ **or** $z_{\text{cur}} > z_{\text{obj}}$ **do**
    for all nodes i in $\mathcal{N}$ **do**
        $\boldsymbol{z}_i^{\text{old}} \leftarrow \boldsymbol{z}_i$
        transmit entries σ_{ij} and $\tilde{\sigma}_{ji}$ to neighbors $j \in \mathcal{N}_i$
        receive entries σ_{ji} and $\tilde{\sigma}_{ij}$ from neighbors $j \in \mathcal{N}_i$
        $\sigma_{ij} \leftarrow \tilde{\sigma}_{ij}$ and $\tilde{\sigma}_{ji} \leftarrow \sigma_{ji}$ ▷ apply $\mathcal{Q}_2$
        $\boldsymbol{x}_i \leftarrow \mathcal{P}_1(\boldsymbol{z}_i/\sqrt{\rho})$ ▷ solve problem (2.33)
        $\boldsymbol{z}_i \leftarrow 2\sqrt{\rho}\,\boldsymbol{x}_i - \boldsymbol{z}_i$ ▷ complete $\mathcal{Q}_1$
        $\boldsymbol{z}_i \leftarrow (1-q)\boldsymbol{z}_i^{\text{old}} + q\boldsymbol{z}_i$ ▷ combine with $\boldsymbol{z}_i^{\text{old}}$
        add $\boldsymbol{z}_i$ to buffer Z
        $\boldsymbol{z}_i \leftarrow A2DR(\mathrm{Z})$ ▷ Anderson acceleration [70]
    end for
    $f_{\text{cur}} \leftarrow J(\boldsymbol{x})$
    $z_{\text{cur}} \leftarrow \|\boldsymbol{z} - \boldsymbol{z}^{\text{old}}\|$
end while
locally allocate workload at slot $k = 1$ using $\boldsymbol{x}_i$

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. 2.2, 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 $\bar{b}_i$.

The simulation environment is entirely realized in Python 3 and built from scratch for this work.

Evaluation metrics

Jain's fairness index. To assess the load balancing capability of job scheduling solutions, we use Jain's fairness index,

$$F(\varphi) = \frac{\left(\sum_{i\in\mathcal{M}} \varphi_i\right)^2}{|\mathcal{M}| \sum_{i\in\mathcal{M}} \varphi_i^2}\,, \tag{2.37}$$

where $\varphi = \{\varphi_i\}_{i\in\mathcal{M}}$ collects the servers load factors, see (2.15), 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

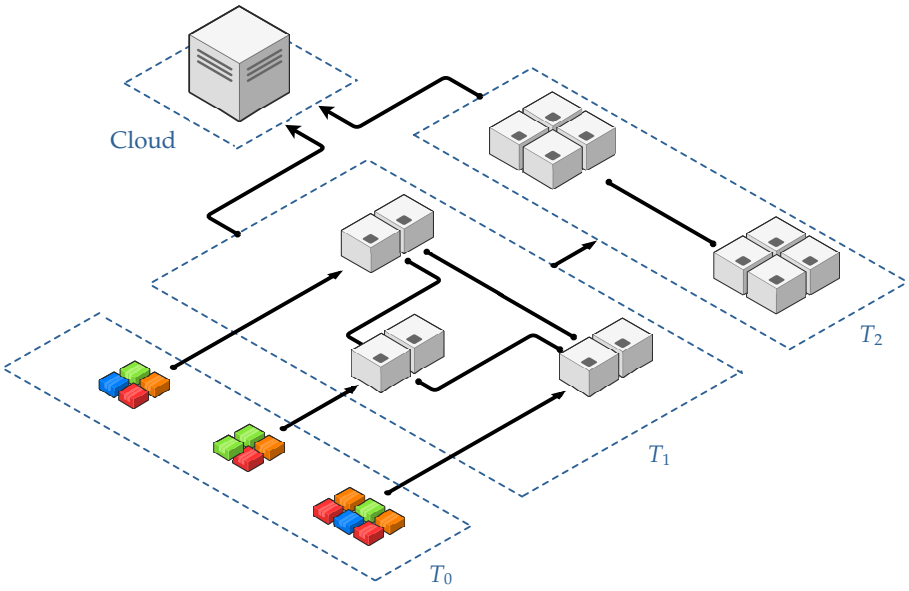


Figure 2.3: Three-tier network. The three tiers T_0, T_1 and T_2 comprise devices generating workload (e.g., IoT nodes and mobile terminals), edge servers, and more powerful edge servers, respectively. Besides, edge servers can also fall back to the cloud computing facility.

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 $\eta_w = \mu_1 W_{\text{edge}}/(\mu_1 W_{\text{edge}} + \mu_2 W_{\text{cloud}})$ weights 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 allocation fairness, the total efficiency metric is defined as

$$\eta = \eta_e \times \eta_w \times F(\varphi)\,. \tag{2.38}$$

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 [37]. 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. It 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,

$$D_i = \frac{\tau_i}{T}. \tag{2.39}$$

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

Table 2.2: 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 $\bar{b}_2$	$2\bar{b}_1$

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.

2.6.2 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. 2.4: the proposed distributed optimization framework of Algorithm 2.3 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 2.6.1. 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. 2.4a. 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 ($\varepsilon \geq 0.7$). This is not attainable with 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. 2.4d, 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 much more intensive use of cloud computing resources starting from $\varepsilon = 0.5$, when T_2 servers are only 40% full. This corresponds to 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 $\varepsilon = 0.7$. 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. 2.4e, is smaller for the proposed algorithm (see the range $\varepsilon \in [0.5, 0.8]$). From Fig. 2.4b, we further see that the amount of energy harvested suffices to keep the battery level to 100% until $\varepsilon = 0.4$, irrespective 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., $\varepsilon \leq 0.5$), 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 (2.37) is plotted in Fig. 2.4c. As it 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 a very low load. With the myopic scheme, the performance slightly degrades, dropping considerably in the range $\varepsilon \in (0, 0.3]$. A further substantial drop is observed with the heuristic.

These facts directly reflect on the global system efficiency η (see Fig. 2.4f), 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. 2.4c), and it remains highest as ε increases, as MPC sends less workload to the cloud facility (Fig. 2.4d).

Figs. 2.5a and 2.5b show the energy traded with the power grid (respectively, sold and purchased). MPC significantly reduces the amount of energy injected into the power grid concerning myopic and heuristic strategies. At low ε, e.g., around $\varepsilon = 0.3$, 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. 2.5a. Also, MPC buys less energy from the power grid, going from 1 kJ/slot (heuristic) to 0.6 kJ/slot (MPC), see Fig. 2.5b. This reflects more efficient management of harvested energy resources by MPC, resulting in a reduction of 50% in the energy traded with the grid. Beyond $\varepsilon \approx 0.6$, 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

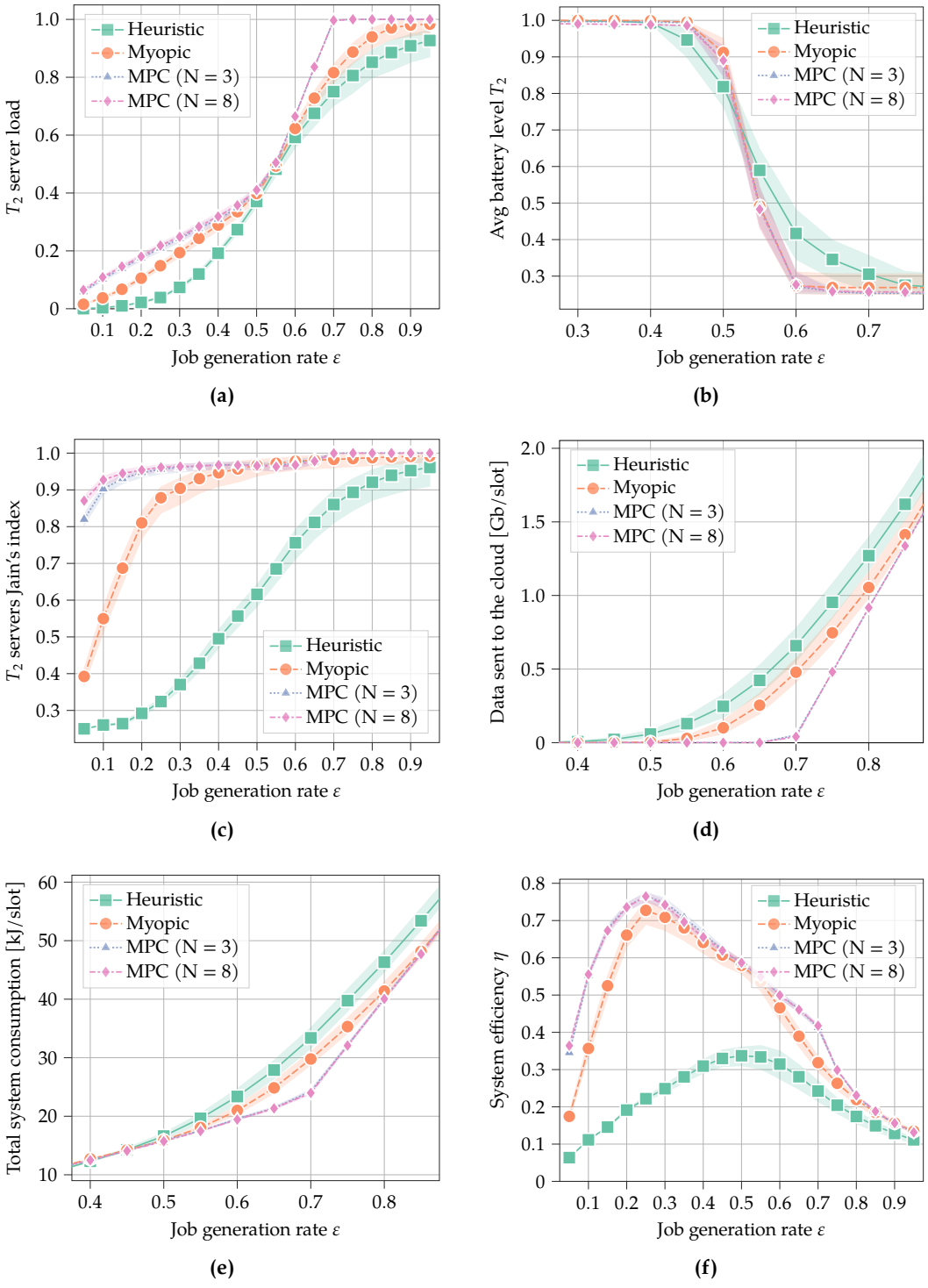
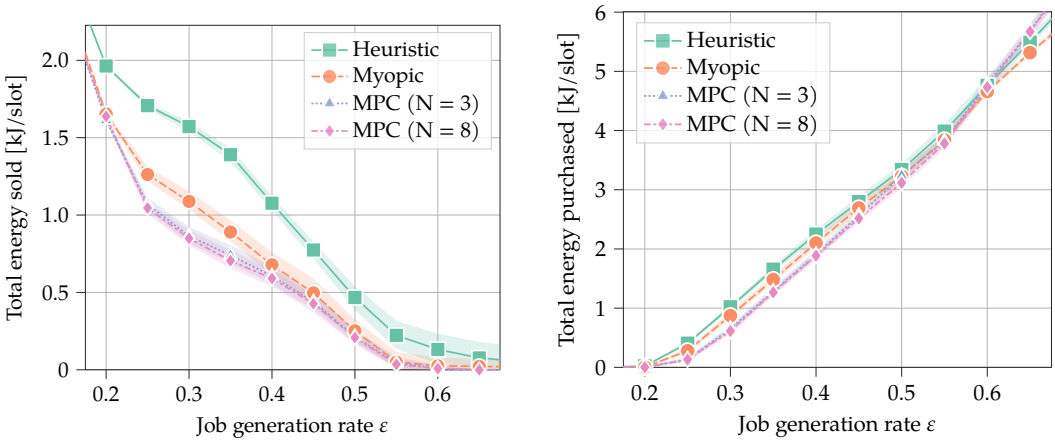


Figure 2.4: Main system features as a function of the job generation rate. In these plots, the quadratic cost function is used for comparison with the benchmark heuristic.

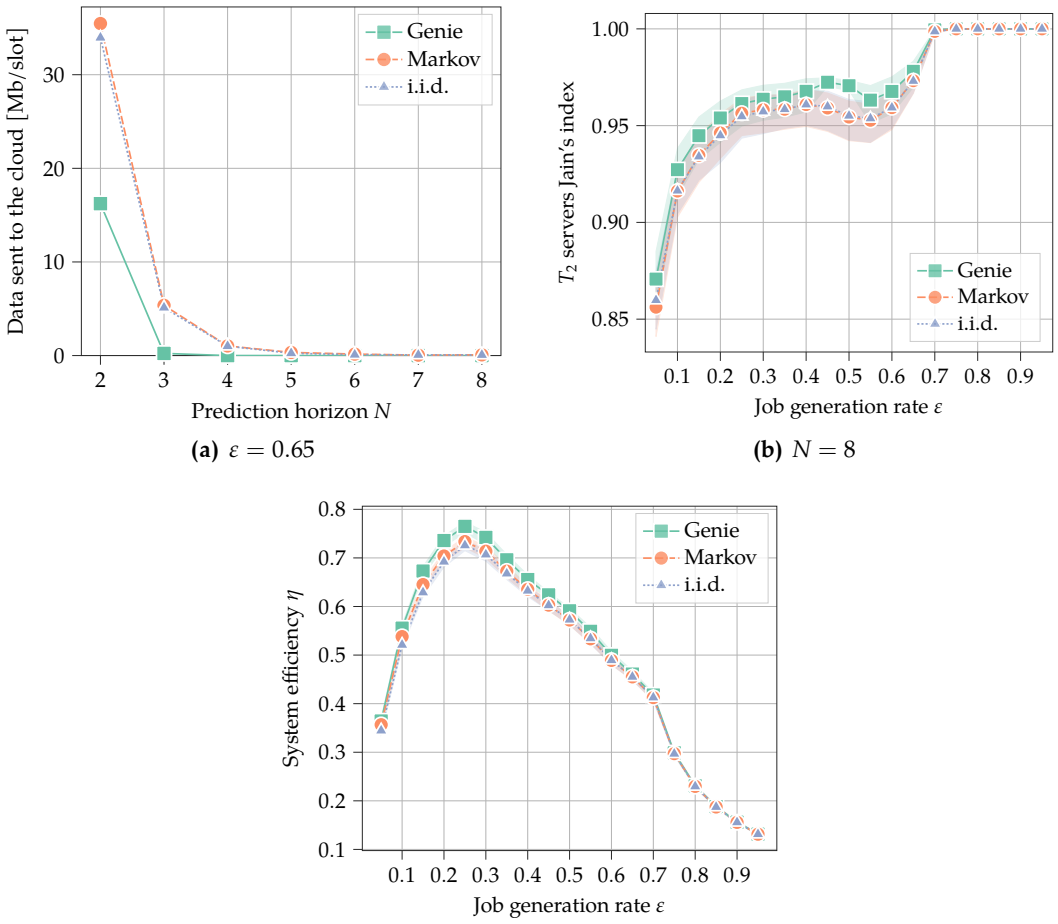


(a) Sold **(b)** Purchased

Figure 2.5: Average energy traded with the power grid.

to their poorer allocation of computing resources and send more workload to the cloud facility (see Fig. 2.4d). When the amount of energy harvested is insufficient, e.g., due to the intermittent energy generation from renewables, MPC automatically resorts to using the energy available from the power grid. As long as the computational capabilities at the edge are sufficient to manage the computing demand, no degradation in the computing performance is observed. On the other hand, the amount of energy drained from the power grid will correspondingly increase.

In Fig. 2.6, 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 2.3.4. Fig. 2.6a 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. 2.6b, 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. 2.6c): although the difference is negligible, there is a slight advantage in using Markov chains at low generation rates.



(a) $\varepsilon = 0.65$

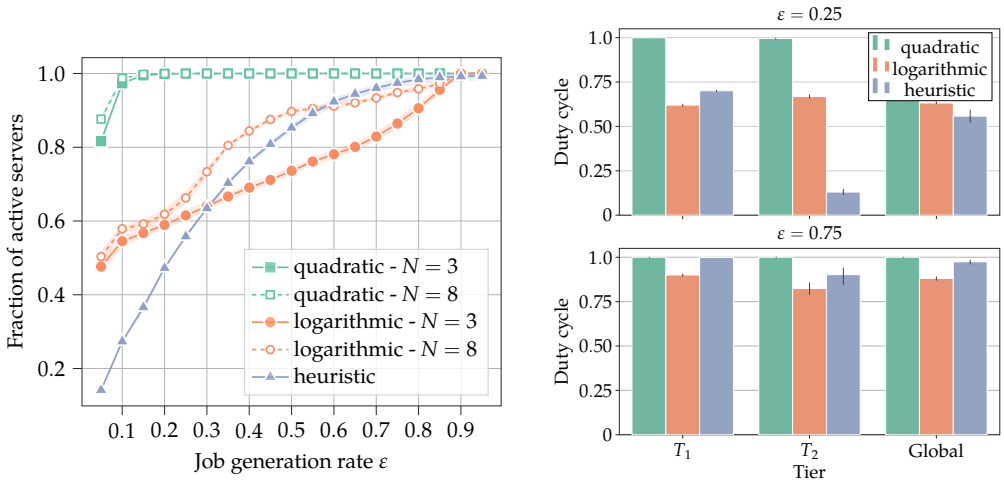
(b) $N = 8$

(c) $N = 8$

Figure 2.6: 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.

2.6.3 Load balancing *vs* consolidation

We now assess the role of the two cost functions of Section 2.4.4. 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. 2.7, we see that the quadratic cost promotes load balancing, while the logarithmic one and the heuristic scheme both induce server consolidation. Specifically, in Fig. 2.7a 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 $\varepsilon < 0.3$, the proposed



(a) $M = 16$ **(b)** $N = 3, M = 16$

Figure 2.7: Comparison of the proposed optimization schemes as per the consolidation metrics.

heuristic achieves the best results in terms of server consolidation. This holds 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 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. 2.7b ($\varepsilon = 0.25$) 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.

2.7 Conclusion

In this chapter, 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 is beneficial concerning a heuristic strategy, and 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 pur-

sued, 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.

3

Energy-efficient and mobility-aware container migration in urban environment

3.1 Introduction

The future of mobile networks is not only concerned with faster and more reliable wireless connections. The rapid digitalization of society [3] comes with a need to expedite the service provisioning time, demanding support for *computation-intensive* and *delay-sensitive* users' applications. Often, these applications cannot be executed on the end devices due to memory and energy scarcity, nor on the network cloud due to a consequent surge in the Internet traffic and excessive delays. These facts lead to the introduction of the MEC paradigm, entailing the de-location of computation services at the mobile network edge, by empowering the evolved Node B (eNB) sites with adequate computing facilities, referred to as MEHs. With MEC, a user can offload intensive computing jobs to a MEH, thus considerably reducing the communication delays with respect to cloud services. Spurred by the high potential of such innovation, the European Telecommunications Standards Institute (ETSI) is extensively working on the standardization of interoperable MEC architectures [71], along with their integration with fifth-generation (5G) – and beyond – mobile networks [72].

In this chapter, we consider an Internet of Vehicles (IoV) scenario, where the network users are 5G – or beyond 5G – enabled vehicles requiring communication and computing support [73], [74]. According to [3], among M2M communications, connected cars is the vertical with the highest expected compound annual growth rate (30%) until at least 2023. Moreover, one of the key challenges in an IoV context is ensuring computing service continuity as the vehicles move away from their serving MEH [75]. This requires implementing online policies to decide whether to move the entity executing the service on a MEH that is closer to the user or to complete the

computation where it started. In the former case, the user spends less energy to communicate with the MEH, but resources are spent by the network due to the migration process, both in terms of energy and time. As for the latter, standard network procedures [76] ensure that the user remains connected to the serving MEH, thus guaranteeing the delivery of the computation result, at the cost of higher latency.

Chapter contribution. We propose Energy-Aware job Scheduling at the Edge (EASE), a *proactive* approach to select the most suitable allocation of computing resources considering energy, memory, and computation constraints. In the envisioned scenario, eNBs (MEHs) are connected to the power grid and empowered with PVs, which provide green energy that can be exploited without additional costs. Vehicle mobility predictions are leveraged to estimate the best sites where the users' computing jobs can be allocated, accounting for network and users' requirements. To the best of our knowledge, this is the first attempt to design a complete framework for the *energy efficient* scheduling of computing jobs over MEHs networks, by exploiting *mobility aware* procedures. The devised system provides job schedules that minimize the carbon footprint on the network side – for the computation and communication services – subject to job latency and mobility constraints. The job scheduling policy consists of two phases, the former is independently and locally executed at the eNBs (MEHs), while the latter is implemented as a decentralized consensus process. In the first phase, each MEH leverages estimates of the renewable (cost-free) energy, the computational power, and the memory available within a prediction window to decide upon the optimal local amount of workload to be executed, subject to users' mobility and delay constraints. Each MEH also identifies the jobs that should be migrated to neighboring MEHs, as belonging to vehicles that are approaching the border of their current serving cell. The mobility predictor developed in [18] is used to determine the desired workload to transfer to each neighboring MEH. Then, in the second phase, the MEHs collectively reach an agreement on the amount of workload to exchange to reduce the overall energy expenditure, while guaranteeing adequate QoS to the end-users: an approximated integer solution for jobs migration is derived through a consensus algorithm followed by a rounding step, using mobility predictions to make job migration decisions.

EASE is evaluated in a real-world scenario emulated through the "simulation of urban mobility" (SUMO) software, considering the vehicular mobility traces for the city of Cologne, and dense city-wide deployment of 5G eNBs with MEC functionalities. Numerical results reveal that the developed allocation strategy significantly reduces the carbon footprint of the edge network, with an increasing gain over heuristic strategies when the available green energy is scarce. At the same time, it properly allocates workload to the processing units according to their specific computing power, by delivering better QoS to the users with respect to heuristic solutions and meeting delay constraints. When possible, service migrations also follow the UE during handovers, i.e., services are migrated to the MEH that is closest to the UE after the handover event.

The present work brings the following innovations.

- The problem of computation service continuity is solved in a holistic way, designing EASE, a complete framework for users' job scheduling and migration within the MEHs of a mobile edge network with distributed renewable energy resources. The main objective is to reduce

the carbon footprint of the computing network by using renewable energy resources to the maximum extent.

- A two-step approach for job location management and migration is devised, splitting the problem into local and distributed phases. With it, MEHs take advantage of user mobility information (and forecasting) to reduce the energy expenditure of the edge network.
- For the distributed phase, a consensus strategy is designed to make migration decisions and solved in closed form by exploiting a dual ascent algorithm. Upon reaching a consensus, an original strategy is put forward to obtain an approximated solution for workload and memory management at the MEHs.

Concerning what presented in Chapter 2, this work i) employs similar techniques of MPC to schedule the local processing of workload at each eNB and ii) takes advantage of the study regarding different statistical predictors. Nonetheless, the working pipeline proposed here splits the local execution scheduling and the distributed offloading decision, resulting in a more lightweight decentralized phase.

The related work is analyzed in the next Section 3.2, whereas the solution workflow is presented in Section 3.3, where we also detail the remaining sections of the chapter.

3.2 Related Work

The resource allocation problem in a MEC scenario with *static* users is extensively addressed in the literature. Among the most recent works, in [77] the authors present a job scheduler for containers management at the MEHs, to reduce the network carbon footprint. In [78], [79], the task offloading is optimized from a user perspective, minimizing the task completion time and the related energy expenditure. However, as these approaches consider static users, they are not suitable for IoV scenarios. Specifically, for IoV, mobility management is a key aspect towards effective implementation of MEC assisted networks [80]. In this chapter, we devise EASE, a scheduling algorithm to guarantee service continuity in MEC assisted IoV networks, by properly allocating computation services based on the network energy distribution and the mobility of the users. Moreover, EASE is specifically designed to reduce the carbon footprint of MEC assisted networks, by considering facilities empowered with renewable energy sources in addition to the supply from the power grid. Note that the user's computation task allocation requires both i) to decide the MEH where to place the job together with the workload to be executed based on the available resources and ii) to trigger service handovers based on the user mobility and energy availability predictions. In fact, computation service handovers entail not only the exchange of control messages but also the migration of the data associated with the specific job under execution. The users' requests are served at a so-called serving MEH through the instantiation of a virtual entity – either a VM or a container – empowered with adequate memory and computing resources to satisfy the service requirements [81]. Therefore, when a computing service handover is triggered, the virtual entity must be transferred to the target MEH, and computation must be restored from

	Objective	Network energy			Network computing resources	Users mobility			
		Migration cost	Computing cost	Carbon footprint		Paths planning	Users distribution	Previously visited cells	Velocity or trajectory
[88]	energy	✓			✓	✓			
[89]	latency					✓			
[90]	energy	✓							✓
[91]	latency				✓			✓	
[92]	latency				✓			✓	
[93]	energy				✓			✓	
[18]	energy	✓			✓			✓	✓
[94]	latency				✓		✓		
EASE	energy	✓	✓	✓	✓			✓	✓

Table 3.1: Summary of the minimization objective quantities and the MEH system aspects considered by EASE and the proactive computing service migration approaches in the literature.

the point where the previous serving MEH stopped. This poses several issues associated with the job latency constraints and the network energy migration costs. A paper addressing the latency challenge, and proposing strategies to reduce the migration time is [82]. The main focus is on *how* to migrate the virtual entity, by defining protocols to transfer the container/VM from the current location to the target one. Machen et al. [83] propose a layered framework to migrate applications encapsulated either in VMs or containers, showing a reduction in service downtime. The authors of [84] leverage the layered nature of the storage system to reduce the overhead in the container file system synchronization between the serving and the target MEHs. However, these approaches are *reactive*, i.e., the service migration is performed *after* the user has moved to the new MEH site. This results in an unavoidable processing delay due to the time required for the virtual entity re-instantiation at the new MEH [80]. EASE is instead *proactive*, as the virtual entity is migrated *before* the handover event occurs, thus reducing the service interruption time. A quantitative evaluation of the difference in the service downtime between the two approaches can be found in, e.g., [85]–[87], where the authors show that proactive approaches are desirable for time-sensitive applications.

Proactive methods require the MEC orchestrator to know the user's next point of attachment to trigger the migration process in advance. Some recent works in the literature show the effectiveness of this strategy, but i) they fail to provide a complete framework to properly allocate the computing jobs within the network entities while jointly considering the users' mobility and the energy, memory, and computing power constraints, and ii) they rely on a centralized orchestrator that computes the best policy to adopt knowing the state of *all* the network entities. Among them, in [88], the MEC service migration process and the physical route for the user to get to the destination are jointly optimized. The problem is solved through a multi-agent deep reinforcement learning approach to meet the job delay requirements with minimum migration cost and travel time. While the work presented in this chapter forces the vehicle to follow a specific physical path, EASE leaves the decision on the physical route to the user and leverages mobility predictions to place the jobs. In [89], Campolo et al. exploit pre-planned vehicle routes to proactively migrate the MEH container so as to follow the user's movements. In [90], the authors leverage the vehicle velocity and its direction to decide if and where, i.e., to which target MEH the virtual entity should be migrated to reduce the cost of multiple successive service migrations while meeting the jobs' delay constraints. This is obtained through a tradeoff between the energy consumed for migrations and the energy needed to eventually transmit the information through the backhaul links that connect the MEHs for service continuity. However, the strategies in [89], [90] do not

consider the constraints on the MEHs computing power, making the solutions not directly applicable in real-world scenarios. In [91], the authors design a policy to decide whether to migrate the virtual entity to a target MEH – estimated through a mobility predictor based on Markov chains – or to keep the job execution on the serving MEH where it was initiated, reallocating the service in case the MEH capacity is exceeded. In [92], the authors use mobility estimates, obtained using a convolutional neural network, to migrate the computation services through a recursive procedure based on genetic algorithms. However, the mobility predictor developed in [91], [92] only considers the sequence of the user's previously visited cells without leveraging the mobility pattern followed by the user within the current radio cell: this fails to precisely capture real-world mobility patterns, as shown in [18]. Moreover, these articles are concerned with minimizing the computing service latency, i.e., energy aspects are not considered. A different approach is presented in [93], where the user's virtual entity is replicated to multiple neighboring MEHs before the handover event occurs, considering the MEHs capacity. The authors suggest using mobility estimates to place the replicas, but leave this for future study. Again, the energy aspect is not considered. These issues are addressed in [18], where the authors integrate accurate predictions – based on the actual trajectory of the user within the eNB coverage area – into a VM replication strategy, to reduce the network energy consumption. However, while the authors show the impact of the MEH computing power on the risk of service discontinuity, they do not introduce a strategy to address this problem.

The above-referenced methods are not concerned with finding the proper allocation of computing jobs when they are offloaded from the user to the network (the service is first placed on the closest MEH). In this respect, Rago et al. [94] use predictions on the distribution of the number of users attached to the different eNBs and estimates of the task requests to proactively allocate jobs on the available MEHs considering computing power constraints. The proposed strategy does not address service migrations and is mainly concerned with minimizing the latency while energy consumption is not considered.

We emphasize that [18], [88]–[94] assume that all MEHs are attached to the power grid for continuous energy provisioning. This makes these approaches not suitable for the scenario considered in this chapter, where we target the reduction of the network carbon footprint in the presence of renewable energy. This aspect was considered in [9], where the authors study the problem of managing the energy coming from renewable sources to minimize the energy drained from the power grid. In [9], MPC is used to jointly allocate the local resources and to obtain offloading decisions toward other servers. Instead, EASE uses MPC to control the local processing only and to obtain an average estimate of future resource availability. In this way, EASE allows reducing the complexity of the solution with respect to the distributed approach in [9] as discussed in Section 3.6.4. Moreover, unlike what we do with EASE, user mobility was not considered [9]. Table 3.1 summarizes the key aspects considered in the previous literature.

In the present work, we propose EASE, an energy- and mobility-aware, distributed, and proactive scheduling framework for computing jobs allocation and virtual entity migration, with the objective of minimizing the carbon footprint of the MEH network. EASE is the first approach that jointly considers all these aspects in addressing the complex problem of efficiently managing

MEC empowered IoV networks. This is achieved by combining local policies with a decentralized consensus algorithm, thus obviating the need for an orchestrator. To show the impact of EASE on the network carbon footprint, we compare the obtained results with the service migration approach in [18] as, using the same mobility predictor, allows revealing the advantages of EASE. Moreover, we implemented three heuristic schemes to approach service migration as presented in [95], i.e., i) never migrate the service ("keep"), ii) always migrate the service when a handover occurs ("migrate"), and iii) define a threshold on a performance metric to decide whether to migrate or not the service ("threshold").

3.3 EASE overview

The network setup consists of an urban environment covered by a set $\mathcal{N}$ of eNBs, each co-located with a MEH. $\mathcal{V}$ represents the set of vehicles moving within the city, which are constantly connected to the nearest eNB node (providing communication support). Vehicle $v \in \mathcal{V}$ sends computing job requests to the closest MEH, which can locally execute the required workload or offload it, either partially or in full, to neighboring MEHs. Also, each vehicle can have a single outstanding job instance (being processed) and can generate a single job request at any time slot only if the previous request has been either fully processed or dropped by the serving MEH. For this reason, in the following analysis, we will interchangeably identify a vehicle with the associated outstanding job to be computed. The set of neighboring eNBs to eNB i is denoted by $\mathcal{N}_i$. Jobs are executed through the instantiation of containers, which reserve the required computing and memory resources. Here, containers are favored over VMs due to their lower memory footprint, which permits a faster migration process – a desirable feature in the considered scenario [82]. Jobs that are being executed on one MEH but associated with vehicles that are about to leave the eNB/MEH coverage area are assessed by the migration controller. The latter decides whether to migrate their execution to another (target) MEH or to finish it locally and send the processing result to the vehicle in a multi-hop fashion (from the old to the new serving eNB). eNBs are equipped with energy harvesting PV devices, whose collected energy is managed by the system. We assume that eNBs are also connected to the power grid as relying only upon harvested energy would be risky due to its intermittent nature; so energy can be drained from the grid when the incoming green energy is scarce or surplus energy can be injected into the grid. MEHs are batteryless, as batteries are often expensive and need periodic replacement – EASE aims at reducing the carbon footprint of such batteryless eNB/MEH system while meeting memory, processing constraints and accounting for the user mobility.

A high-level diagram of EASE is presented in Figure 3.1, while the diagram of an eNB/MEH node is shown in Fig. 3.2. The scheduler operates according to two optimization phases: 1. a local phase (left of the diagram): a predictive control phase, performed locally at each MEH node, and 2. a distributed phase (right): a collaborative optimization based on distributed consensus (solved via message passing). In phase 1, the MEHs locally control the ongoing computations, estimating the local processing capacity and energy availability within a given prediction horizon. At the same time, the local algorithm assesses the amount of workload that should be migrated

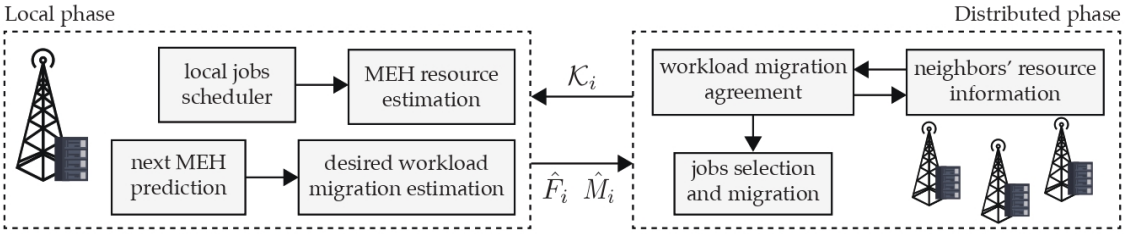


Figure 3.1: High-level diagram of EASE. The local steps (left) provide the resource and the desired workload migration estimates for each MEH in isolation. The distributed algorithm (right) allows MEHs to reach a consensus on the jobs allocation and trigger their migration.

("desired workload migration estimation") to the neighboring MEH nodes, predicts the availability of local resources ("MEH resource estimation"), and accounts for mobility estimates ("next MEH prediction"), i.e., the vehicle that generated the job request is about to hand over to a neighboring radio cell.

With phase 2, taking the desired workload to be migrated from phase 1 as input ("neighbors' resource information"), the MEHs collectively reach an agreement ("workload migration agreement") about *how many* and *which* jobs are to be migrated, as well as about the target MEH for their migration ("job selection and migration").

After phase 2), each node updates its local state equations with the new jobs generated by the vehicles under coverage and those received from the neighbors, and goes back to phase 1).

In the remainder, the system model is presented in Section 3.4. The problem formulation for the optimal scheduling is detailed in Section 3.5. The final scheduling solution, composed of the two phases (local and distributed) is presented in Section 3.6. The performance assessment is reported in Section 3.7 and final remarks are provided in Section 3.8.

3.4 System model

Next, we detail the mathematical models for computing and communication services, along with the statistical processes involved in the envisioned scenario and the system constraints. Time t is discrete and evolves according to slots of fixed duration τ, i.e., $t = 0, \tau, 2\tau, \ldots$. The mathematical notation is summarized in Table 3.2.

3.4.1 Computation and communication models

Computing job parameters. At time t, each job k served by MEH i is characterized by the triplet $(I_{i,k}(t), D_{i,k}(t), S_{i,k}(t))$, where i) $I_{i,k}(t)$ is the residual job intensity, expressed in CPU cycles, ii) $D_{i,k}(t)$ is the residual (hard) execution deadline, in seconds, i.e., the time still available to execute the job, and iii) $S_{i,k}(t)$ is the remaining data to be processed, in bits. As the job is processed by the server,

Symbol	Meaning	Unit
$v \in \mathcal{V}$	vehicle Identifier (ID) and set of vehicles	-
$i \in \mathcal{N}$	eNB/MEH ID and set of eNBs/MEHs	-
$\mathcal{N}_i$ and N_i	set of neighboring nodes of node i and its cardinality $\lvert\mathcal{N}_i\rvert$	-
$k \in \mathcal{K}_i(t)$ and $K_i(t)$	job ID, set of jobs in execution at MEH i at slot t, and its cardinality $\lvert\mathcal{K}_i(t)\rvert$	-
$\hat{\mathcal{K}}_{ij}$ and $\hat{K}_{ij}$	set of jobs running on MEH i with probable next MEH j and its cardinality	-
T	no. of slots in the prediction horizon	-
$t = [0, \ldots, T]$	scheduling time slot index	-
τ	length of a scheduling slot	s
$V_i(t)$ and $C_i(t)$	no. of results to be sent in the coverage area of eNB i at slot t and to be routed through the backhaul network	-
I_k	intensity of job k	cyc.
D_k	deadline of job k	s
S_k	size of job k	bit
p_v and p_ℓ	job generation probability and probability that it is of type ℓ	-
$\boldsymbol{p}_{i,k}(t)$	handover prob vector for vehicle v (job k) at slot t	
$w_{i,k}(t)$	workload of job k processed by MEH i in slot t	cyc.
L	(fixed) size of a container instantiated on a MEH	bit
E_b^{RAN}	energy per bit for eNB-vehicle wireless transmissions	J/bit
E_b^{wired}	energy per bit for eNB-eNB wired transmission	J/bit
σ_s and σ_d	energy per bit for migration at the source (destination) MEH	J/bit
E_s and E_d	(fixed) energy for migration at the source (destination) MEH	J
$E_i^{\text{H}}(t)$	harvested energy available at slot t	J
$P_i^{\text{PV}}(t)$	power supplied by the PV at node i, instant t	W
P_{RAN} and P_{wired}	(fixed) power to keep the wireless (wired) unit switched on	W
P_i^{idle}	(fixed) power to keep the server switched on	W
$N_i^{\text{inc}}(t)$ and $N_i^{\text{out}}(t)$	no. of MEH incoming (outgoing) jobs at slot t	-
F_i	maximum computational power of server i	W
M_i	maximum amount of RAM available at server i	bit
$\bar{w}_{ij}$	desired intensity requested by MEH i to neighbor j	cyc./s
$\bar{m}_{ji}$	memory space requested by MEH i to neighbor j	bit
$\hat{P}_i^{\text{H}}$	residual green power at node i after the local scheduling	W
$\hat{F}_i$	residual computing power at node i after local scheduling	W
$\hat{M}_i$	residual RAM memory at node i after the local scheduling	bit
o_{ij}	optimal amount of MEH i processing load to offload to j	cyc./s
$\tilde{o}_{ji}$	optimal processing load to be received at MEH i from j	cyc./s

Table 3.2: Summary of the symbols used throughout the chapter. "cyc." stands for "CPU cycles".

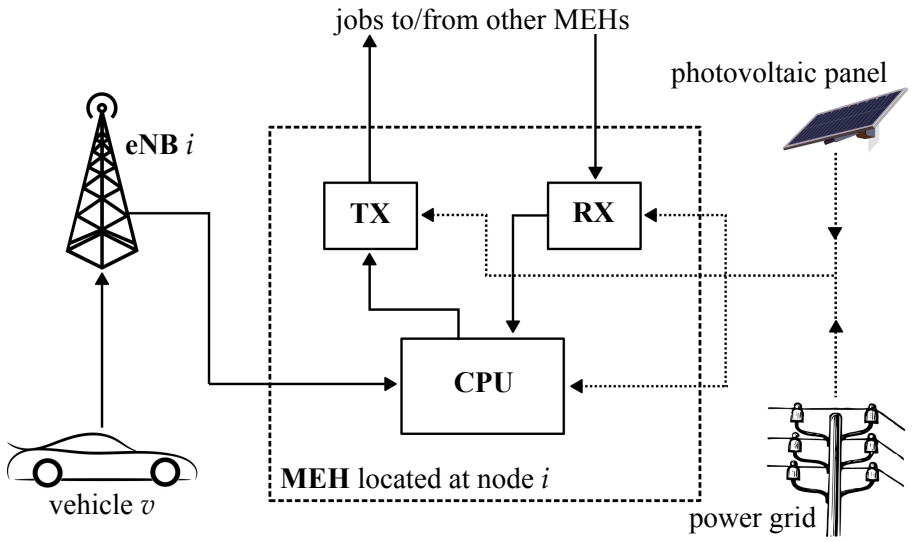


Figure 3.2: eNB/MEH node. Job requests arrive from connected vehicles v moving within the eNB coverage area. Containers handling the execution of the jobs are created at the serving MEH, and possibly migrated to other MEHs in case the associated vehicles exit the eNB coverage area.

the intensity, deadline, and data size decrease according to

$$I_{i,k}(t+\tau) = I_{i,k}(t) - w_{i,k}(t)\,, \tag{3.1}$$

$$D_{i,k}(t+\tau) = D_{i,k}(t) - \tau\,, \tag{3.2}$$

$$S_{i,k}(t+\tau) = S_{i,k}(t) - \frac{S_{i,k}(0)}{I_{i,k}(0)} w_{i,k}(t)\,, \tag{3.3}$$

where $w_{i,k}(t)$ is the amount of workload (CPU cycles) belonging to job k and processed by MEH i in slot t, $S_{i,k}(0)$ represents the initial job size (bits), whereas $I_{i,k}(0)$ is the total number of CPU cycles required to fully process the job. Eq. (3.3) means that the amount of data that is still to be processed decreases linearly with the amount of workload allotted to a job, irrespective of how the workload is distributed in time. Note that (3.1) makes it possible to rewrite (3.3) as

$$S_{i,k}(t) = \frac{S_{i,k}(0)}{I_{i,k}(0)} I_{i,k}(t). \tag{3.4}$$

Communication models. For the 5G wireless links between the eNBs and the vehicles we adopt i) the massive-MIMO energy consumption model of [96], and ii) the mm-wave – 28 GHz – urban NLoS channel model of [97]. Specifically, from [96] the following system parameters are obtained: i) the power needed to keep the wireless unit switched on (fixed circuit power consumption), P_{RAN}, ii) the energy required per transmitted bit via wireless links, E_b^{RAN}, iii) the fixed wired circuit power consumption, P_{wired}, iv) the energy expenditure for the wired backhaul links connecting the eNBs, E_b^{wired}. Note that the vehicles' energy utilization is not involved in the scheduling and,

in turn, only the energy consumption at the eNB side is considered. The model in [97] is used for the vehicle-eNB association.

Container migration model. The migration of a container requires the hosting MEH to spend energy to freeze the status of the virtual entity and prepare the data to be sent to the target MEH for the correct re-instantiation. Hence, the target MEH has to spend energy to create the new virtual entity using the received information. The energy expenditure on the two sides consists of [98]: i) a contribution proportional to the size of the migration data, through the parameters σ_s and σ_d respectively, plus ii) a fixed energy contribution, equal to E_s for the source MEH and E_d for the target one, respectively. Additionally, the source spends some energy to transmit the data over the wired channel E_b^{wired}. Overall, it holds

$$E_{\text{source}}^{\text{migr}}(t) = \sigma_s L + E_b^{\text{wired}} S_k(t) + E_s, \quad \text{and} \tag{3.5}$$

$$E_{\text{dest}}^{\text{migr}}(t) = \sigma_d L + E_d, \tag{3.6}$$

where $S_k(t)$ is the (variable) data size associated with job k, and L is the (fixed) container size. According to [89], we account for a service downtime of T_k^{migr} when migrating the entities. In turn, T_k^{migr} seconds are additionally removed from the job's deadline $D_k(t)$ at every migration occurrence. Note that the delay associated with wired transmissions is negligible as compared to the service downtime.

3.4.2 Statistical processes

Energy harvesting model. We refer to $P_i^{\text{PV}}(t)$ as the power supplied by the PV co-located with eNB/MEH i at instant t and that varies from a minimum of $P_{\min}^{\text{PV}}$ to a maximum of $P_{\max}^{\text{PV}}$. Accounting for the power required to keep the server (P_i^{idle}) and the communication channels (P_{RAN} and P_{wired}) switched on, and the fixed amount of energy required for the container migration, the harvested energy available at eNB/MEH i for computations and data transmissions at time slot t is

$$\begin{aligned} E_i^{\text{H}}(t) = &\left(P_i^{\text{PV}}(t) - P_{\text{RAN}} - P_{\text{wired}} - P_i^{\text{idle}}\right)\tau + \\ &- N_i^{\text{inc}}(t)\left(\sigma_d L + E_d\right) - N_i^{\text{out}}(t)\left[\left(\sigma_s + E_b^{\text{wired}}\right)L + E_s\right] \end{aligned} \tag{3.7}$$

where $N_i^{\text{inc}}(t)$ and $N_i^{\text{out}}(t)$ are the known number of MEH incoming and outgoing jobs at MEH i and time t, which are scheduled at the previous step $t - \tau$. The terms in Eqs. (3.5)-(3.6) that depend on the data size $S_{i,k}(t)$ are not considered in $E_i^{\text{H}}(t)$ as they will be integrated into the optimization function (see Eq. (3.17)). Note that being $E_i^{\text{H}}(t)$ a difference between the harvested energy and that required to deliver the services, its value can be negative. $E_i^{\text{H}}(t)$ is known for the current slot t only. However, the developed MPC framework also needs estimates for $[E_i^H(t+\tau), \ldots, E_i^H(t+\tau(T-1))]$, within the time window $t+\tau, \ldots, t+\tau(T-1)$, where T is the prediction horizon. These estimates are computed by forecasting the time-dependent quantities in (3.7): future values of $P_i^{\text{PV}}(t+\cdot)$ are estimated using a Gaussian r.v. with average P_{PV} and standard deviation σ_{PV}, estimates for the number of incoming $N_i^{\text{inc}}(t+\cdot)$ and outgoing $N_i^{\text{out}}(t+\cdot)$ jobs at

eNB i in slot t are obtained considering the vehicles in the external annulus of the eNB's coverage area. Finally, P_i^{idle} depends on the specific MEH characteristics at eNB i, as specified in Section 3.7.

Jobs types and arrival model. Three job types are considered for the numerical results of Section 3.7, having different intensities, deadlines, and data sizes and identified through the index $\ell = \{1, 2, 3\}$. Every job type is associated with a generation triplet (I^ℓ, D^ℓ, S^ℓ), and a generation probability p_ℓ. Each vehicle $v \in \mathcal{V}$ can submit at most one computing job at a time to the network facilities so that a bijective mapping vehicle-job ID can be derived. Once a job is finished or expired, the vehicle submits a new job to the MEH with probability p_v at each slot. This parameter is tuned in the simulations. Also in this case, for predictive optimization, an estimate for the future incoming jobs is needed. For this purpose, a circular buffer containing the values of $I_{i,k}/D_{i,k}$ of the newly generated jobs is kept. A fixed estimate of the average of the last W seconds is used to predict the incoming traffic. In [9], the authors verified that even simple predictors are still effective with MPC if T is large enough.

Handover probabilities. Each job k is associated with a probability vector that depends on the position of the vehicle v requesting the service. Being i the serving eNB for vehicle v, we define $\boldsymbol{p}_{i,k}(t)$ as the N_i-dimensional vector containing the probabilities that vehicle v will hand over to any of the $|\mathcal{N}_i| = N_i$ neighboring radio cells, i.e., $\boldsymbol{p}_{i,k}(t) = [p_{i1,k}(t), p_{i1,k}(t), \ldots, p_{iN_i,k}(t)]$, with $\sum_j p_{ij,k} = 1$. Vector $\boldsymbol{p}_{i,k}(t)$ is updated every time a new trajectory sample is available for the associated vehicle v, either inside the same cell or in a new cell after performing the handover.

3.4.3 System constraints

The set $\mathcal{K}_i(t)$, with cardinality$K_i(t) = |\mathcal{K}_i(t)|$, collects the jobs being executed at time slot t at MEH i. The following systems constraints apply

Processing capacity. Indicating with F_i the maximum computing power of server i – expressed in CPU cycles per second – the following inequality on the sum of the workloads holds

$$\frac{1}{\tau} \sum_{k=1}^{K_i(t)} w_{i,k}(t) \leq F_i \,. \tag{3.8}$$

Storage capacity. Being M_i [bits] the maximum amount of RAM available at server i, the sum of the data sizes $S_{i,k}(t)$ of all the active jobs at MEH i must obey

$$\sum_{k=1}^{K_i(t)} S_{i,k}(t) \leq M_i \,. \tag{3.9}$$

Job execution time. In case the deadline of job k, $D_{i,k}(t)$, expires in the current time slot t, the job must be processed entirely and immediately at server i and cannot be further migrated, i.e.,

$$w_{i,k}(t) = I_{i,k}(t) \qquad \text{if } D_{i,k}(t) \leq \tau \,. \tag{3.10}$$

This guarantees the timely delivery of the computation result to the requesting vehicle, avoiding the outcome becoming useless. As Eqs. (3.8)-(3.10) may not be jointly satisfied, in the following we will relax Eq. (3.8).

Workload conservation. Finally, note that, in general, the inequalities

$$0 \leq w_{i,k}(t) \leq I_{i,k}(t), \ \forall\, i \in \mathcal{N}, \ \forall\, k \in \mathcal{K}_i(t), \ \forall\, t \tag{3.11}$$

must always hold, because of the workload conservation principle.

3.5 Problem formulation

Here we formulate the optimization problems concerning the 1) local and 2) distributed scheduling phases introduced in Section 3.3. As shown in Figure 3.1, the local and distributed schedulings are run in parallel as distinct tasks that exchange information.

3.5.1 Local phase: Local controller and resources estimation

Each MEH $i \in \mathcal{N}$ estimates $w_{i,k}(t)$ for every job $k \in \mathcal{K}_i(t)$ to be executed at time t: in the analysis, $w_{i,k}(t)$ stands for the optimal fraction of computing intensity $I_{i,k}(t)$ to be locally executed at time slot t for the hosted job k. We define vectors $\boldsymbol{w}_i(t)$, $\boldsymbol{I}_i(t)$ and $\boldsymbol{D}_i(t)$ respectively collecting $w_{i,k}(t)$, $I_{i,k}(t)$ and $D_{i,k}(t)$ for all $k \in \mathcal{K}_i(t)$. As for the energy spent to transmit the processing results back to the vehicles, $V_i(t)\, E_b^{\text{RAN}}$ is the (per bit) energy cost of sending the results to the $V_i(t)$ vehicles in the wireless coverage area, while $C_i(t)\, E_b^{\text{wired}}$ is the energy cost entailed in routing the $C_i(t)$ jobs that are completed at node i and that have to be routed via the backhaul links to reach the corresponding user (vehicle). R_k is the size of the processing result of job k, and q_i^{proc} is the energy cost of processing a unit of workload.

Given these quantities, we define two local (at node i) functions $f_i(\cdot)$ and $g_i(\cdot)$, as follows.

$$f_i(\boldsymbol{w}_i; V_i, C_i, E_i^{\text{H}}) = q_i^{\text{proc}} \mathbf{1}^T \boldsymbol{w}_i(t) + V_i(t)\, E_b^{\text{RAN}} R_k + C_i(t)\, E_b^{\text{wired}} R_k - E_i^{\text{H}}(t)\,, \tag{3.12}$$

$$g_i(\boldsymbol{I}_i(t); \boldsymbol{D}_i(t)) = \sum_{k=1}^{K_i(t)} \left(\frac{I_{i,k}(t)}{D_{i,k}(t)} \right)^2 . \tag{3.13}$$

$f_i(\cdot)$ quantifies the difference between the total energy expenditure at node i in slot t (due to processing and communications processes) and the energy that is locally harvested at this node. Hence, $-f(\boldsymbol{w}_i; \cdot)$ represents the residual cost-free energy available for the migration process in the distributed phase. Minimizing $f_i(\cdot)$ corresponds to maximizing the local energy available at the node. $g_i(\cdot)$ represents the residual processing cost, which is proportional to $(I_{i,k}/D_{i,k})^2$. Minimizing $g_i(\cdot)$ forces the node to execute the jobs, especially prioritizing those with high intensity and whose deadline is about to expire. Note also that, due to Eq. (3.1), $I_{i,k}(t)$ depends on the optimization variable $w_{i,k}$ at previous time slots.

Considering a forecast optimization window of T slots into the future, and letting $t = 0$ be the current time slot, the local cost function at node i over the whole time horizon is formulated by combining $f_i(\cdot)$ and $g_i(\cdot)$, as

$$J_i\left(\mathcal{W}_i, \mathcal{I}_i; \mathcal{D}_i, \boldsymbol{V}_i, \boldsymbol{C}_i, \boldsymbol{E}_i^{\mathrm{H}}\right) = \gamma \sum_{t=0}^{T-1} g_i(\boldsymbol{I}_i(t); \boldsymbol{D}_i(t)) + \sum_{t=0}^{T-1} \max\{f_i(\boldsymbol{w}_i; \cdot), 0\}^2, \tag{3.14}$$

where $\mathcal{W}_i$, $\mathcal{I}_i$ and $\mathcal{D}_i$ represent the stacks of vectors $\boldsymbol{w}_i(t)$, $\boldsymbol{I}_i(t)$ and $\boldsymbol{D}_i(t)$ over the considered horizon T, respectively, while $\boldsymbol{V}_i$, $\boldsymbol{C}_i$ and $\boldsymbol{E}_i^{\mathrm{H}}$ are the vectors collecting $V_i(t)$, $C_i(t)$ and $E_i^{\mathrm{H}}(t)$ for $t \in \{0, \tau, \ldots, \tau(T-1)\}$. The coefficient $\gamma > 0$ is used to balance the processing state cost term ($g_i(\cdot)$) with respect to the energy cost ($f_i(\cdot)$).

Remark 1. *From a physical perspective, the processing energy consumption is not necessarily a quadratic function, but it varies based on the specific computing architecture* [99]. *A quadratic function for $f_i(\cdot)$ was chosen, as it promotes smoothness of the controller in the transitions from one slot to the next one, and has the same curvature order of the processing state cost $g_i(\cdot)$. Also, the* $\max\{\cdot\}$ *function is used to make the cost positive only when $f_i(\cdot) > 0$, i.e., the renewable energy is fully used and the node has to resort to the power grid.*

Next, the cost function in Eq. (3.14) is modified through the addition of a penalty term proportional to two non-negative auxiliary variables $\delta_i(t) = [\delta_{F_i}(t), \delta_{M_i}(t)]$, to ensure that the problem does not become infeasible when resources are scarce. Therefore, rewriting the constraints (3.8) and (3.9), we define for each MEH the following local problem at node i,

$$\begin{aligned} P_i^{\mathrm{loc}}: \quad \min_{\mathcal{W}_i, \delta_i} \quad & J_i\left(\mathcal{W}_i, \delta_i; \cdot\right) + \sum_{t=0}^{T-1} \boldsymbol{c}_i^T \delta_i(t) \\ \text{s.t.} \quad & (3.1) \text{ - } (3.3),\ (3.10),\ (3.11), \\ & \frac{1}{\tau} \sum_{k \in \mathcal{K}_i} w_{i,k}(t) \leq F_i + \delta_{F_i}(t), \\ & \sum_{k \in \mathcal{K}_i} S_{i,k}(t) \leq M_i + \delta_{M_i}(t), \\ & \delta_{F_i}(t) \geq 0,\ \delta_{M_i}(t) \geq 0, \end{aligned} \tag{3.15}$$

where $\boldsymbol{c}_i = [c_{F_i}, c_{M_i}]$ is the vector collecting the coefficients weighting the penalty variables, with $c_{F_i}, c_{M_i} > 0$. By solving (3.15), each MEH obtains the optimal control $\boldsymbol{w}_i(0)$ which is implemented in the current time step.

3.5.2 Distributed phase: Workload migration agreement

From (3.15), each server estimates its future energy and processing resources. Specifically, let $\hat{P}_i^{\mathrm{H}}$ be the residual available green power, possibly negative if the grid support is sought, $\hat{F}_i$, and $\hat{M}_i$ be the residual computational power, and RAM memory at node i, respectively. Note that,

since constraints (3.8) and (3.9) are relaxed in (3.15), $\hat{F}_i$ and $\hat{M}_i$ can be negative. These estimates are obtained by averaging the values over the prediction horizon, excluding the current instant $t = 0$. Due to this averaging operation, while in (3.15) we deal with energy expenditures, in the following we refer to power quantities.

The migration task presents itself as a combinatorial Mixed Integer Programming (MIP) problem, which is non-convex and is generally difficult to solve in a distributed fashion. Thus, we use heuristics to derive approximated solutions. In this chapter, the popular *relax and round* method is used, which consists in solving the convex counterpart of the original problem and rounding the result to a feasible solution afterward. The reason for this choice is that it allows tackling the problem in a distributed fashion via *message passing*, solving the continuous form problem exactly to the optimum. Other approaches would have required a centralized solution or the design of a heuristic inspired by the optimization objective.

Based on the handover probability vector $\boldsymbol{p}_{i,k}$ presented in Section 3.4.2, each MEH determines the average resource demand requested from its neighbors in the migration process. Specifically, the CPU cycles per second and memory space that are requested from neighbor j are

$$\bar{w}_{ij} = \sum_{k \in \hat{\mathcal{K}}_{ij}} \frac{I_{i,k}}{D_{i,k}}\,, \quad \text{and} \quad \bar{m}_{ij} = \sum_{k \in \hat{\mathcal{K}}_{ij}} S_{i,k}\,, \tag{3.16}$$

respectively, where $\hat{\mathcal{K}}_{ij}$ contains the set of jobs that are currently running at server i, associated with vehicles that are about to leave the coverage area of the co-located eNB i and whose most probable next eNB is co-located with MEH j. With $\bar{\boldsymbol{w}}_i = [\bar{w}_{i1}, \ldots, \bar{w}_{iN_i}]$ we denote the vector collecting the *desired* processing intensity per second to be sent to each of the N_i neighbors of MEH i, computed via (3.16). We also introduce the new optimization variables $\boldsymbol{o}_i = [o_{i1}, \ldots, o_{iN_i}]$ and $\tilde{\boldsymbol{o}}_i = [\tilde{o}_{1i}, \ldots, \tilde{o}_{N_i i}]$ representing the optimal total amount of processing load to be sent to and to be received from each neighbor, respectively. The deviation from the desired $\bar{\boldsymbol{w}}_i$ to be migrated is penalized with the l_2-norm $\|\bar{\boldsymbol{w}}_i - \boldsymbol{o}_i\|^2$, and the migration cost is defined as

$$\Gamma_i\left(\boldsymbol{o}_i, \tilde{\boldsymbol{o}}_i; \bar{\boldsymbol{w}}_i, \hat{P}_i^{\mathrm{H}}\right) = \max\left\{\left(q_i^{\mathrm{tx}} - q_i^{\mathrm{proc}}\right)\mathbf{1}^T\boldsymbol{o}_i + \left(q_i^{\mathrm{rx}} + q_i^{\mathrm{proc}}\right)\mathbf{1}^T\tilde{\boldsymbol{o}}_i - \hat{P}_i^{\mathrm{H}}, 0\right\} + \rho\|\boldsymbol{o}_i - \bar{\boldsymbol{w}}_i\|^2\,, \tag{3.17}$$

where q_i^{proc}, q_i^{tx} and q_i^{rx} are the processing, transmission and reception costs of server i (expressed as powers), respectively. The $\max\{\cdot\}$ term accounts for the power that would be drained from the power grid to migrate the jobs, whereas the quadratic term encodes the fact that the optimal $\boldsymbol{o}_i$ should be as close as possible to the desired $\bar{\boldsymbol{w}}_i$ – this corresponds to moving the jobs to the next serving eNB. Finally, $\rho > 0$ is a weight balancing the importance of the two cost terms. Note that minimizing Eq. (3.17) returns a solution $\boldsymbol{o}_i$ that matches vector $\bar{\boldsymbol{w}}_i$ if the residual harvested power is sufficient and the constraints are satisfied. Specifically, as system constraint we consider the following variation of (3.8) and (3.9), introducing a variable $\hat{\delta}_i \geq 0$, as follows,

$$\sum_{j \in \mathcal{N}_i}\left(\tilde{o}_{ji} - o_{ij}\right) \leq \min\{\hat{F}_i, \xi_{M_i}\hat{M}_i\} + \hat{\delta}_i, \qquad \forall\, i \in \mathcal{N}. \tag{3.18}$$

Remark 2. *The meaning of* (3.18) *is that the workload surplus that server i has during the following time steps, i.e., the* incoming *workload minus the* outgoing *one, should satisfy the average* (*long-term*) *power* ($\hat{F}_i$) *and memory* ($\hat{M}_i$) *availability at node i. The coefficient* ξ_{M_i} *relates the memory availability to the residual computational power. This follows from the assumption of direct proportionality between the data size* S_k *and the processed workload* w_k.

Since the general goal is to minimize the energy drained network-wide from the power grid, a cost function that represents the global welfare and at that at the same time is amenable to a distributed solution is the sum

$$\Gamma\left(\boldsymbol{o}, \tilde{\boldsymbol{o}}, \hat{\delta}; \bar{\boldsymbol{w}}, \hat{\boldsymbol{P}}^{\mathrm{H}}\right) = \sum_{i \in \mathcal{N}} \left[\Gamma_i\left(\boldsymbol{o}_i, \tilde{\boldsymbol{o}}_i; \bar{\boldsymbol{w}}_i, \hat{P}_i^{\mathrm{H}}\right) + \hat{c}_i \hat{\delta}_i^2\right], \tag{3.19}$$

where $\hat{c}_i > 0$ is the cost coefficient associated with the penalty term $\hat{\delta}_i^2$. This leads to the constrained optimization problem

$$\begin{aligned} P^{\mathrm{glob}}: \quad \min_{\boldsymbol{o}, \tilde{\boldsymbol{o}}, \hat{\delta}} \quad & \Gamma\left(\boldsymbol{o}, \tilde{\boldsymbol{o}}, \hat{\delta}; \bar{\boldsymbol{w}}, \hat{\boldsymbol{P}}^{\mathrm{H}}\right) \\ \text{s.t.} \quad & \boldsymbol{o}, \tilde{\boldsymbol{o}}, \hat{\delta} \geq 0, \ (3.18), \\ & o_{ij} = \tilde{o}_{ij} \ \forall\, i, j, \end{aligned} \tag{3.20}$$

with $\boldsymbol{o}$, $\tilde{\boldsymbol{o}}$, $\hat{\delta}$, $\bar{\boldsymbol{w}}$ and $\hat{\boldsymbol{P}}^{\mathrm{H}}$ are vectors collecting $\boldsymbol{o}_i$, $\tilde{\boldsymbol{o}}_i$, $\hat{\delta}_i$, $\bar{\boldsymbol{w}}_i$ and $\hat{P}_i^{\mathrm{H}}$ respectively, for all the MEHs $i \in \mathcal{N}$. The equality $o_{ij} = \tilde{o}_{ij}$ is called *consensus constraint* and ensures that the amount of workload exiting node i and directed to j equals the one that j expects to receive from i.

3.5.3 On the interaction between local and distributed phases

The local problem (3.15) is used to schedule the amount of workload w_i (CPU cycles) that is to be executed locally at each MEH in the current time slot t. Since the solution is predictive, it uses future memory availability ($\hat{M}_i$) and residual computational power ($\hat{F}_i$) estimates to set the global problem constraints (3.18). Thanks to the global problem (3.20) an agreement is reached on which jobs are to be migrated and where. The solution $\boldsymbol{o}_i$ of the global problem is utilized to move workload across the MEHs: this entails an update of sets $\mathcal{K}_i(t+1)$ containing the jobs that are assigned to MEH i at the next time slot $t+1$. The optimization keeps iterating between local and distributed phases.

3.6 Problem solution

3.6.1 Phase 1: local MPC solution

At each MEH, the local MPC problem of (3.15) is solved over the whole horizon T [43]. MPC uses the *receding horizon* technique, which consists of solving the given problem within a predic-

tion window of size T, applying the optimal computed control only for the current time step $t = 0$, moving forward the optimization window by one time slot (τ seconds) and repeating the procedure. In this way, the controller progressively adapts to new observations and estimates of the exogenous processes. Also, at any given instant, MEH i computes the optimal policy throughout the whole horizon of T slots, but only $\boldsymbol{w}_i(0)$ is applied as the control action. The exogenous processes are the future jobs and the harvested energy availability, see Section 3.4.2.

3.6.2 Phase 2a: distributed workload migration

In the following, the scheduling slot index t is omitted in the interest of readability. Eq. (3.20) is a *consensus* problem, i.e., it entails reaching an agreement on the value of some variables among multiple agents in a distributed system. In our context, the MEHs must agree on the amount of processing load to exchange among each other. A way to solve this problem – written as the sum of separable convex cost functions – is via the *dual ascent* algorithm [100]. Given a generic cost function $\psi(\boldsymbol{x})$, its *Lagrangian* is defined as

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{z}) = \psi(\boldsymbol{x}) + \boldsymbol{z}^T (A\boldsymbol{x} - \boldsymbol{d}), \tag{3.21}$$

where $\boldsymbol{z}$ are the Lagrange multipliers associated with the constraints $A\boldsymbol{x} = \boldsymbol{d}$. The dual ascent solves the problem by iteratively i) minimizing $\mathcal{L}(\boldsymbol{x}, \boldsymbol{z})$ with respect to $\boldsymbol{x}$ (primal step), and ii) updating the value of $\boldsymbol{z}$ (dual step). To formalize the solution of problem (3.20) via dual ascent, we split the local cost functions (3.17) as

$$\begin{aligned} \tilde{\Gamma}_i \left(\boldsymbol{o}_i, \tilde{\boldsymbol{o}}_i, \hat{\delta} \right) = &\max \left\{ \left(q_i^{\text{tx}} - q_i^{\text{proc}} \right) \mathbf{1}^T \boldsymbol{o}_i + \left(q_i^{\text{rx}} + q_i^{\text{proc}} \right) \mathbf{1}^T \tilde{\boldsymbol{o}}_i - \hat{P}_i^{\text{H}}, 0 \right\} + \\ &+ \frac{\rho}{2} \| \boldsymbol{o}_i - \bar{\boldsymbol{w}}_i \|^2 + \frac{\rho}{2} \| \tilde{\boldsymbol{o}}_i - \tilde{\boldsymbol{w}}_i \|^2 + \hat{c}_i \hat{\delta}_i^2, \end{aligned} \tag{3.22}$$

exploiting the fact that $o_{ij} = \tilde{o}_{ij}$, and defining $\tilde{\boldsymbol{w}}_i = \{ \bar{w}_{ji} \,|\, j \in \mathcal{N}_i \}$. Intuitively, node i is responsible for half of the quadratic cost from its neighbors and for half of its own local cost. For compactness, let $\boldsymbol{x} = \{ \boldsymbol{x}_i = [\boldsymbol{o}_i, \tilde{\boldsymbol{o}}_i, \hat{\delta}_i], \, \forall \, i \in \mathcal{N} \}$ be the global optimization variable, $\boldsymbol{b}_i = [\bar{\boldsymbol{w}}_i, \tilde{\boldsymbol{w}}_i, 0]$ the tracking target vector, and $\boldsymbol{q}_i = [\boldsymbol{q}_i^{\text{tx}} - \boldsymbol{q}_i^{\text{proc}}, \boldsymbol{q}_i^{\text{rx}} + \boldsymbol{q}_i^{\text{proc}}, 0]$ the linear costs vector. Moreover, we define matrix $Q_i = I_{2N_i+1} \boldsymbol{m}_i$, with $\boldsymbol{m}_i = [\frac{\rho}{2}, \ldots \frac{\rho}{2}, \hat{c}_i]$, and the global block diagonal matrix Q, collecting each Q_i on the diagonal. With these definitions, problem (3.20) can be expressed in the following form

$$\min_{\boldsymbol{x}} \quad \sum_{i \in \mathcal{N}} \left(\| \boldsymbol{x}_i - \boldsymbol{b}_i \|_{Q_i}^2 + \max \left\{ \boldsymbol{q}_i^T \boldsymbol{x}_i - \hat{P}_i^{\text{H}}, 0 \right\} \right) \tag{3.23}$$

$$\text{s.t.} \quad A_1 \boldsymbol{x} \leq \boldsymbol{d}, \tag{3.24}$$

$$A_2 \boldsymbol{x} = \boldsymbol{0}, \tag{3.25}$$

where $\| \boldsymbol{x} \|_Q^2 = \boldsymbol{x}^T Q \boldsymbol{x}$. The inequalities (3.24) collect (3.18) and the non-negativity constraints $\boldsymbol{o}, \tilde{\boldsymbol{o}}$, $\hat{\delta} \geq 0$, while the equalities (3.25) correspond to the consensus constraints $o_{ij} = \tilde{o}_{ij}$, $\forall i \in \mathcal{N}$, $j \in \mathcal{N}_i$. Here, matrices A_1 and A_2 are used to select the concerned variables, whereas $\boldsymbol{d} = \{ \boldsymbol{d}_i = [\min\{\hat{F}_i,$

Algorithm 3.1 Dual ascent algorithm solving problem (3.20)

1: $\boldsymbol{x}^+ = \text{argmin}_{\boldsymbol{x}}\ \mathcal{L}(\boldsymbol{x}; \boldsymbol{y}, \boldsymbol{z})$ ▷ primal
2: $\boldsymbol{y}^+ = \max\left\{\boldsymbol{y} + \alpha_{\boldsymbol{y}}\left(A_1 \boldsymbol{x}^+ - \boldsymbol{d}\right), 0\right\}$ ▷ dual (ineq.)
3: $\boldsymbol{z}^+ = \boldsymbol{z} + \alpha_{\boldsymbol{z}} A_2 \boldsymbol{x}^+$ ▷ dual (eq.)

$\xi_{M_i}\hat{M}_i\}, \mathbf{0}] \mid i \in \mathcal{N}\}$. We can now write the Lagrangian as

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \sum_{i \in \mathcal{N}} \tilde{\Gamma}_i\left(\boldsymbol{x}_i; \boldsymbol{b}_i, \hat{P}_i^{\mathrm{H}}\right) + \boldsymbol{y}^T\left(A_1 \boldsymbol{x} - \boldsymbol{d}\right) + \boldsymbol{z}^T A_2 \boldsymbol{x}\,, \tag{3.26}$$

where $\boldsymbol{y} = \{\boldsymbol{y}_i = [\lambda_i, \gamma_i, \tilde{\gamma}_i, \hat{\phi}_i] \mid i \in \mathcal{N}\}$ are the Lagrange multipliers associated with the inequality constraints (3.24), and $\boldsymbol{z} = \{\boldsymbol{z}_i = \mu_i \mid i \in \mathcal{N}\}$ are the multipliers associated with equalities (3.25). Specifically, the Lagrange multipliers λ_i refer to constraints (3.18), $\gamma_i = \{\gamma_{ij}\}$, $\tilde{\gamma}_i = \{\tilde{\gamma}_{ji}\}$ and $\hat{\phi}_i$ to $\boldsymbol{o}_i \geq 0$, $\tilde{\boldsymbol{o}}_i \geq 0$, and $\hat{\delta}_i \geq 0$, respectively, and $\mu_i = \{\mu_{ij}\}$ to $o_{ij} = \tilde{o}_{ij}$, for every server $i \in \mathcal{N}$, and $j \in \mathcal{N}_i$. Using the $^+$ sign to denote the update at the following iteration, we detail in Algorithm 3.1 the dual ascent procedure that solves the problem

$$\inf_{\boldsymbol{x}} \sup_{\boldsymbol{y} \geq 0, \boldsymbol{z}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})\,. \tag{3.27}$$

The dual update requires in this case two different forms, depending on whether the constraint is an equality or an inequality one. Inequality constraints may actually be inactive, and the associated Lagrange multipliers would be null in this case. The parameters $\alpha_{\boldsymbol{y}}$ and $\alpha_{\boldsymbol{z}}$ in the algorithm tune the stability and the convergence speed. The presented compact version of the dual ascent translates into the following local procedure, from a server perspective. Defining vectors $\tilde{\mu}_i = \{\mu_{ji}\}$ and $\bar{\boldsymbol{o}}_i = \{\tilde{o}_{ij}\}$ to collect those variables that are kept in memory by the neighborhoods of i, the local Lagrangian at node i is

$$\begin{aligned}\mathcal{L}_i\left(\boldsymbol{x}_i; \bar{\boldsymbol{w}}_i, \tilde{\boldsymbol{w}}_i, \hat{P}_i^{\mathrm{H}}, \nu_i\right) = \tilde{\Gamma}_i\left(\boldsymbol{o}_i, \tilde{\boldsymbol{o}}_i; \bar{\boldsymbol{w}}_i, \tilde{\boldsymbol{w}}_i, \hat{P}_i^{\mathrm{H}}\right) + \lambda_i\left[\mathbf{1}^T(\tilde{\boldsymbol{o}}_i - \boldsymbol{o}_i) - \hat{\delta}_i\right] + \\ - \gamma_i^T \boldsymbol{o}_i - \tilde{\gamma}_i^T \tilde{\boldsymbol{o}}_i + \mu_i^T \boldsymbol{o}_i - \tilde{\mu}_i^T \tilde{\boldsymbol{o}}_i - \hat{\phi}_i \hat{\delta}_i,\end{aligned} \tag{3.28}$$

with $\boldsymbol{x}_i = [\boldsymbol{o}_i^+, \tilde{\boldsymbol{o}}_i^+, \hat{\delta}_i]$ and $\nu_i = [\lambda_i \mathbf{1}, \phi_i, \gamma_i, \tilde{\gamma}_i, \mu_i, \tilde{\mu}_i]$ to collect the Lagrange multipliers. The local procedure is presented in Algorithm 3.2, where a fixed step size α is assumed.

Note that, to minimize the Lagrangian in the primal step at line 2, server i not only needs its own Lagrange multipliers, but also the introduced $\tilde{\mu}_i$, which collects the μ_{ji} of neighbors $j \in \mathcal{N}_i$. Therefore, node i must first receive these multipliers from the neighborhood. Also, while updating μ_i in the dual step at line 9, $\bar{\boldsymbol{o}}_i^+$ is needed, which collects the $\tilde{o}_{ij}^+$ variables kept by the neighborhood of i, and which are to be received after the computation of j's primal step ($\forall j \in \mathcal{N}_i$). Hence, this amounts to two communication rounds among neighbors per dual ascent iteration. The dual updates are computationally inexpensive, whereas the primal step requires solving a local convex subproblem, which is complicated by the $\max\{\cdot\}$ operator in the cost function (3.22). Eventually, note that additional communication is required at the beginning of the procedure, to inform the

Algorithm 3.2 Dual ascent from a server perspective

1: receive $\tilde{\mu}_i = \{\mu_{ji}\}$ from the neighbors
2: $[\boldsymbol{o}_i^+, \tilde{\boldsymbol{o}}_i^+, \hat{\delta}_i] = \text{argmin}_{\boldsymbol{x}_i} \; \mathcal{L}_i\left(\boldsymbol{x}_i; \bar{\boldsymbol{w}}_i, \tilde{\boldsymbol{w}}_i, \hat{P}_i^{\text{H}}, \nu_i\right)$
3: send $\tilde{o}_{ji}^+$ to the corresponding neighbor j
4: $\lambda_i^+ = \max\left\{\lambda_i + \alpha\left(\sum_{j\in\mathcal{N}_i}\left(\tilde{o}_{ji}^+ - o_{ij}^+\right) - \hat{F}_i\right), 0\right\}$
5: $\hat{\phi}_i^+ = \max\left\{\hat{\phi}_i - \alpha\,\hat{\delta}_i^+, 0\right\}$
6: $\gamma_i^+ = \max\left\{\gamma_i - \alpha\,\boldsymbol{o}_i^+, 0\right\}$
7: $\tilde{\gamma}_i^+ = \max\left\{\tilde{\gamma}_i - \alpha\,\tilde{\boldsymbol{o}}_i^+, 0\right\}$
8: receive $\bar{\boldsymbol{o}}_i^+ = \{\tilde{o}_{ij}^+\}$ from the neighbors
9: $\mu_i^+ = \mu_i + \alpha\left(\boldsymbol{o}_i^+ - \bar{\boldsymbol{o}}_i^+\right)$
10: send μ_{ij}^+ to the corresponding neighbor j

neighborhood about the values of $\tilde{\boldsymbol{w}}_i$.

Solution to the primal step (line 2). The solution of the local primal subproblems is computed in closed form, distinguishing three cases. We consider the local primal subproblems in compact form with variables $\boldsymbol{x}_i$, and collect the Lagrange multipliers of (3.28) in $\nu_i = [\lambda_i \mathbf{1}, \phi_i, \gamma_i, \tilde{\gamma}_i, \mu_i, \tilde{\mu}_i]$, with associated variables selection matrix A_i. We split $\mathcal{L}_i\left(\boldsymbol{x}_i; \cdot\right) = u_i\left(\boldsymbol{x}_i\right) + h_i\left(\boldsymbol{x}_i\right)$, so that

$$u_i\left(\boldsymbol{x}_i\right) = \|\boldsymbol{x}_i - \boldsymbol{b}_i\|_{Q_i}^2 + \nu_i^T A_i \boldsymbol{x}_i, \tag{3.29}$$

$$h_i\left(\boldsymbol{x}_i\right) = \boldsymbol{q}_i^T \boldsymbol{x}_i - \hat{P}_i^{\text{H}}. \tag{3.30}$$

Proposition 1. *The solution of the primal step of problem* (3.20) *is computed as one of the mutually exclusive cases*

i) $\boldsymbol{x}_i^+ = \text{argmin}_{\boldsymbol{x}_i} u_i\left(\boldsymbol{x}_i\right)$, *if* $h_i\left(\boldsymbol{x}_i^+\right) \leq 0$, *or*

ii) $\boldsymbol{x}_i^+ = \text{argmin}_{\boldsymbol{x}_i} u_i\left(\boldsymbol{x}_i\right) + h_i\left(\boldsymbol{x}_i\right)$, *if* $h_i\left(\boldsymbol{x}_i^+\right) > 0$, *or*

iii) $\boldsymbol{x}_i^+ = \text{argmin}_{\boldsymbol{x}_i} u_i\left(\boldsymbol{x}_i\right)$, *s.t.* $h_i\left(\boldsymbol{x}_i\right) = 0$.

Proof. i) and ii) correspond to the cases where the max$\{\cdot\}$ operator in (3.22) is replaced by 0 or $h_i(\boldsymbol{x}_i)$, respectively. Once the optimum is computed, the feasibility check must be done: if the minimum lies in the feasible region, the solution is accepted. However, it can also be that these two optima are both infeasible: in this case, the optimal solution must lie on the plane $h_i(\boldsymbol{x}_i) = 0$, and a constrained problem has to be solved (case iii)). □

Remark 3. *It is impossible that both solutions i) and ii) are feasible, otherwise the convex function* (3.22) *would have two minima, which is absurd due to its convexity.*

The solutions for each of the cases of Proposition 1 are now given in the following result.

Proposition 2. *Consider the three cases of Proposition 1. Their closed-form optimal solutions are expressed as*

$$
\begin{aligned}
&i)\ \ \boldsymbol{x}_i^+ = \boldsymbol{b}_i - \tfrac{1}{2}\, Q_i^{-1} A_i^T \nu_i \\
&ii)\ \ \boldsymbol{x}_i^+ = \boldsymbol{b}_i - \tfrac{1}{2}\, Q_i^{-1} \left(A_i^T \nu_i + \boldsymbol{q}_i\right) \\
&iii)\ \ \boldsymbol{x}_i^+ = \boldsymbol{b}_i - \tfrac{1}{2}\, Q_i^{-1} \left(A_i^T \nu_i + \boldsymbol{q}_i^T \frac{2\, Q_i \left(\boldsymbol{b}_i - \frac{\hat{P}_i^{\mathrm{H}}}{\|\boldsymbol{q}_i\|^2}\boldsymbol{q}_i\right) - A_i^T \nu_i}{\|\boldsymbol{q}_i\|^2} \boldsymbol{q}_i \right)
\end{aligned}
$$

Proof. The proof is straightforward for cases i) and ii): it is sufficient to set the gradient of the function to zero. In the third case, it is necessary to solve the constrained minimization of $u(\boldsymbol{x}_i)$ subject to $h(\boldsymbol{x}_i) = 0$. The Lagrange multipliers method can be used, where the Lagrangian of case iii) is $\mathcal{L}'(\boldsymbol{x}_i, \eta_i) = u(\boldsymbol{x}_i) + \eta_i\, h(\boldsymbol{x}_i)$, and its primal solution is

$$
\inf_{\boldsymbol{x}_i} \sup_{\eta_i} \ \|\boldsymbol{x}_i - \boldsymbol{b}_i\|_{Q_i}^2 + \nu_i^T A_i \boldsymbol{x}_i + \eta_i \left(\boldsymbol{q}_i^T \boldsymbol{x}_i - \hat{P}_i^{\mathrm{H}}\right). \tag{3.31}
$$

The partial derivatives with respect to $\boldsymbol{x}_i$, and η_i are

$$
\begin{aligned}
\frac{\partial \mathcal{L}'(\boldsymbol{x}_i, \eta_i)}{\partial \boldsymbol{x}_i} &= 2\, Q_i \left(\boldsymbol{x}_i - \boldsymbol{b}_i\right) + A_i^T \nu_i + \eta_i\, \boldsymbol{q}_i, \\
\frac{\partial \mathcal{L}'(\boldsymbol{x}_i, \eta_i)}{\partial \eta_i} &= \boldsymbol{q}_i^T \boldsymbol{x}_i - \hat{P}_i^{\mathrm{H}}.
\end{aligned} \tag{3.32}
$$

Setting them to zero, we obtain

$$
\boldsymbol{x}_i = \boldsymbol{b}_i - \frac{1}{2}\left[Q_i^{-1}\left(A_i^T \nu_i + \eta_i\, \boldsymbol{q}_i\right)\right] = \frac{\hat{P}_i^{\mathrm{H}}}{\|\boldsymbol{q}_i\|^2} \boldsymbol{q}_i, \tag{3.33}
$$

from which it is possible to derive the optimal value for the Lagrange multiplier

$$
\eta_i^* = \frac{\boldsymbol{q}_i^T \left[2\, Q_i \left(\boldsymbol{b}_i - \frac{\hat{P}_i^{\mathrm{H}}}{\|\boldsymbol{q}_i\|^2}\boldsymbol{q}_i\right) - A_i^T \nu_i\right]}{\|\boldsymbol{q}_i\|^2}. \tag{3.34}
$$

Now, plugging (3.34) into (3.33) returns the optimal value $\boldsymbol{x}_i^+$ for case iii). □

Remark 4. *For quadratic programs, it is possible to find a condition on the step size α for which the algorithm is ensured to converge. This only depends on the constraint matrices A_1 and A_2, and on the quadratic cost matrix Q defining the curvature. Since these values do not change among the three different primal optimization cases, a common condition can be obtained, i.e.,*

$$
\alpha \leq \frac{2}{\left\| \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} Q^{-1} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}^T \right\|}. \tag{3.35}
$$

Proof. This result can be derived using proposition 2.3.2 of [100]. □

3.6.3 Phase 2b: rounding to a feasible discrete solution

In this section, we show how to compute the actual discrete allocation of jobs by obtaining new variables $\boldsymbol{o}_i^r$, which are the *rounded* versions of the $\boldsymbol{o}_i$ that were previously computed through consensus (see Section 3.6.2). In particular, $\boldsymbol{o}_i$ contains the optimal continuous amount of workload that each MEH would like to send to its neighbors. Instead, its rounded version $\boldsymbol{o}_i^r$ contains a feasible allocation accounting for the fact that the number of jobs and the possible ways of allocating them are discrete.

To compute the new $\boldsymbol{o}_i^r$, as an initial solution, we select the jobs from set $\hat{\mathcal{K}}_{ij}$, whose associated vehicle is about to migrate from eNB site i to j. The rounded $\boldsymbol{o}_i^r$ is thus initially set to $\bar{\boldsymbol{w}}_i$, assuming that the minimizer of the objective function (3.17) is the vector that minimizes the quadratic term. Then, the difference between this guess and the actual optimum obtained from the proposed dual ascent algorithm is computed, $\boldsymbol{o}_i^{\text{diff}}$. For every neighbor j it is now clear whether more workload is to be added to (in case $o_{ij}^{\text{diff}} < 0$) or removed from ($o_{ij}^{\text{diff}} > 0$) the initial guess o_{ij}^r. The jobs that were initially scheduled for migration to node j but that are eventually retained for computation at node i are those minimizing $\|\boldsymbol{o}_i^{\text{diff}}\|_1$. Instead, new jobs are added to the migration list using the prediction vectors $\boldsymbol{p}_{ij}$. In detail, the added jobs are those for which the handover probabilities towards j are maximized. A threshold ε_P is used to approximate the rounded solution, as the continuous optimum $\boldsymbol{o}_i$ will likely not coincide with any possible discrete approximation. The procedure is detailed in Algorithm 3.3.

3.6.4 Additional considerations

Handling pathological cases: Since system constraints are made soft to avoid primal infeasibility, three pathological cases may arise, namely, 1. the optimal processed workload at the current instant exceeds the computational capacity; or 2. the data size for the currently running jobs do not fit the RAM memory; or 3. the deadline expires during the current slot, but the residual intensity is greater than zero. A greedy algorithm is developed to handle all of them. For the first two, the MEH ranks the active jobs through a double ordering criterion, considering as the first ranking criterion the time slot when they expire, and as the second their intensity (or data size). Next, it momentarily pauses the execution of the services starting from the last one in the ordered list, until the resources suffice to proceed. In case 1, when pausing a job m, the amount of processed workload becomes $\sum_{k\in\mathcal{K}_i} w_{i,k} - w_{i,m}$, while in case 2, the data relative to suspended jobs is deleted from the RAM. The number of suspended jobs is the minimum such that the requirements are satisfied. Moreover, in case 1, it is likely that, when a job is suspended, additional computational power becomes available. In such a case, the new computational resources are assigned to the jobs that are closest to their deadline. Case 3 is managed considering the amount of residual intensity $I_{i,k}$. If $I_{i,k}$ is smaller than a threshold ε, then the deadline is extended by a small amount, so that the controller will privilege the execution of the corresponding job in the next slot. In this way, jobs are allowed to finish with a little additional delay (within one slot). If, however, the amount

Algorithm 3.3 Job-neighbor association

Input: mobility pattern predictions matrix P_i; optimal outgoing workload amount $\boldsymbol{o}_i$; set of the jobs $\mathcal{J}_i$ in execution at MEH i; tolerance threshold ε_P.
Output: job-neighbor association sets $\mathcal{Z}_{ij}\ \forall j \in \mathcal{N}_i$; rounded $\boldsymbol{o}_i^r$.
remove jobs $\{k \mid I_{i,k} < \varepsilon \vee D_{i,k} < 2\}$ from $\mathcal{J}_i$
$\boldsymbol{o}_i^r \leftarrow \bar{\boldsymbol{w}}_i$
$\mathcal{Z}_{ij} \leftarrow \hat{\mathcal{K}}_{ij}$
$\mathcal{J}_i \leftarrow \mathcal{J}_i \setminus \bigcup_{j \in \mathcal{N}_i} \hat{\mathcal{K}}_{ij}$
$\boldsymbol{o}_i^{\text{diff}} \leftarrow \boldsymbol{o}_i^r - \boldsymbol{o}_i$ ▷ workload to be adjusted
for all neighbors j in $\mathcal{N}_i$ **do**
    while $o_{ij}^{\text{diff}} > \varepsilon_P$ **do**
        $k \leftarrow$ job of $\mathcal{Z}_{ij}$ minimizing $\left| o_{ij}^{\text{diff}} \right|$
        remove job k from $\mathcal{Z}_{ij}$
        $o_{ij}^r \leftarrow o_{ij}^r - I_{i,k}/D_{i,k}$
        $o_{ij}^{\text{diff}} \leftarrow o_{ij}^{\text{diff}} - I_{i,k}/D_{i,k}$
        add job k to $\mathcal{J}_i$ ▷ make it available for neighbors
    end while
    while $o_{ij}^{\text{diff}} < -\varepsilon_P$ **do**
        take $k \in \mathcal{J}_i \mid k \in \text{argmax}\, \boldsymbol{p}_{ij}$ ▷ most prob. $i \to j$
        add job k to $\mathcal{Z}_{ij}$
        $o_{ij}^r \leftarrow o_{ij}^r + I_{i,k}/D_{i,k}$
        $o_{ij}^{\text{diff}} \leftarrow o_{ij}^{\text{diff}} + I_{i,k}/D_{i,k}$
        mask entry $p_{ij,k}$ ▷ s.t. k is not selected again
    end while
end for

of residual intensity is larger than ε, the job is dropped, i.e., in this case, the algorithm failed to provide an acceptable solution.

Predictions inaccuracies: With the adopted approach, a residual migration suboptimality is still possible also due to prediction errors on the mobility of the users, the service request, and the available local resources. Concerning the mobility prediction, the performance is extensively studied in [18], where the authors compare the mobility predictor also used by EASE with a simpler and less accurate approach based on Markov chains, showing the improvement brought by considering the information on the actual trajectory followed by the users. However, we recall that the main objective of EASE is to reduce the carbon footprint and, in turn, even in the case of precise mobility predictions, the scheduler can decide to place the service in a MEH that is far away from the vehicle, it this leads to better use of the energy resources. For this reason, the eNBs/MEHs are connected via backhaul links that always ensure that the result is sent back to the user. Regarding instead the statistical processes that control the energy availability and the job requests, with EASE we only assume to know the average income over a prediction horizon of some seconds (e.g., with $T = 5, 15$ s). In the previous Chapter 2 the impact of average versus estimated (via Markov chains) or exact knowledge into the future (i.e., a "genie predictor") was assessed, showing that MPC is highly effective even when simple predictors are used.

	HP ProLiant DL 110	Nettrix R620 G40
idle power P_i^{idle}	94 W	110 W
max load power P_i^{max}	299 W	468 W
computational power F_i	3.3 Gflops	7.6 Gflops
RAM memory M_i	64 GB	256 GB

Table 3.3: Servers specifications [99].

	I^ℓ [Gflop]	D^ℓ [s]	S^ℓ [GB]	p_ℓ
type 1	10	20	2	0.4
type 2	16	30	10	0.2
type 3	12	40	0.1	0.4

Table 3.4: Jobs parameters for the simulations.

3.7 Numerical results

EASE is assessed in an emulated environment featuring 5G-enabled vehicles moving within an urban scenario. Mobility traces are obtained with SUMO [101], an open-source traffic simulator to obtain mobility traces around a predefined city road map. For this, we use the "TAPAS Cologne" scenario, which mimics the vehicular traffic within the city of Cologne for a whole day based on the traveling habits of the city dwellers [102]. The scheduling simulator was instead built from scratch using Python 3. The mobile network is composed of 8 eNBs endowed with MEH functionalities, wired connected through optical links. The mobility area is covered with hexagonal cells with an eNB in the center, and with an inter-distance among nodes of 400 m. We generated and collected 24h long SUMO mobility traces with 25 ms granularity, for each of the 8 eNBs in the deployment. The first 15 hours were used to train and validate the mobility prediction algorithm, which is taken from [18], whereas the remaining ones to assess the performance of EASE. For the evaluation, we considered vehicles approaching the edge of the serving eNB coverage area, i.e., that are about to hand over to a new eNB/MEH. With the considered setup, this occurs, on average, when a user is less than 40 meters apart from the radio cell's border. The energy consumption of the MEHs is computed based on the SPECpower benchmark [99]. We selected two different edge computing platforms, namely, an HP ProLiant DL 110 Gen 10 Plus and a Nettrix R620 G40, obtaining two clusters of edge servers with different energy consumption, processing speed, and memory, see Table 3.3. In Table 3.4, we report the jobs intensities, deadlines, data sizes, and generation probabilities, according to the system model of Section 3.4.2. They were chosen to be of interest for a tasks migration purpose (i.e., long enough so that vehicles perform at least one handover) and heterogeneous from both a computational and memory perspective. In this way, it is important also to choose *which* tasks to migrate, e.g., because they have a different size. The other system parameters are listed in Table 3.5.

In the following analysis, the edge energy consumption is evaluated through i) the processing and migration power, averaged across all the MEHs, ii) the energy efficiency, defined as

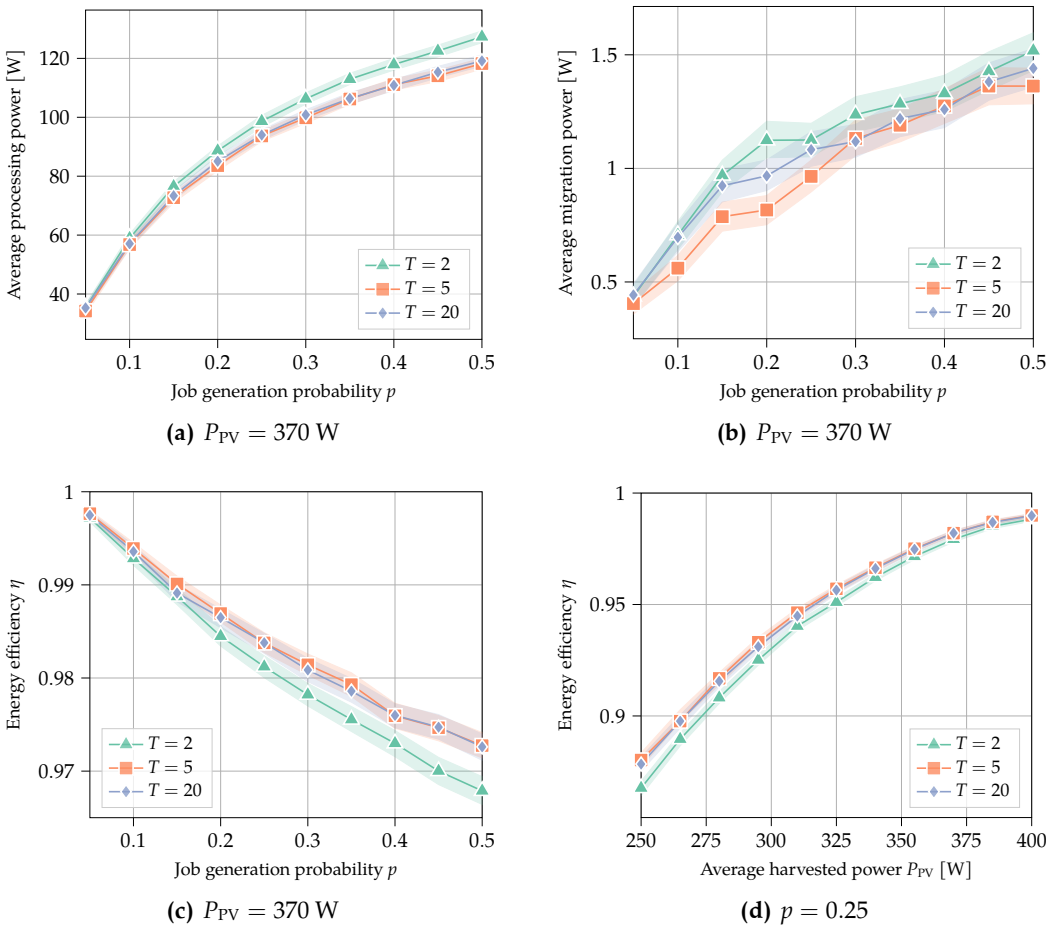
Parameter	Value
number of nodes $\lvert\mathcal{N}\rvert$	8
fixed wireless circuit power consumption P_{RAN}	50.2 W
fixed wired circuit power consumption P_{wired}	20 W
energy per transmitted bit via wireless link E_b^{RAN}	1 nJ/bit
energy per transmitted bit via wired link E_b^{wired}	250 pJ/bit
PV panel minimum power $P_{\text{min}}^{\text{PV}}$	250 W
PV panel maximum power $P_{\text{max}}^{\text{PV}}$	400 W
PV panel average power P_{PV}	370 W
PV panel power std σ_{PV}	10 W
containers' size L	50 MB
weight parameters for L in (3.6) σ_s, σ_d	500 nJ/bit
fixed container migration energy expenditure E_s, E_d	250 mJ
delay associated with wired transmissions T_k^{migr}	2 s
window size to predict incoming traffic W	5 minutes
scheduler time slot τ	3 s
MPC horizon T	$\{2, 5, 20\}$
job generation probability p	0.25
weight of the soft constraint penalty c_i of (3.15)	500
weight of the soft constraint penalty $\hat{c}_i$ of (3.20)	10
state cost γ for Eq. (3.14)	100
weight of the quadratic term of (3.20) ρ	2.5

Table 3.5: Summary of simulation parameters.

$\eta = E_h/E_{\text{tot}}$, i.e., the fraction of harvested (green) energy used over the total energy drained (green plus grid energy), iii) the fraction of executed and finished jobs, and iv) the fraction of jobs finishing in the MEH that is co-located with the eNB serving the vehicle. First, we assess the impact of the prediction window size T on the performance of EASE, then we compare it with the three heuristic migration strategies proposed in [95] (i.e., "keep", "migrate" and "threshold") and the solution of [18], based on Lyapunov optimization and termed thus "lyapunov" (see Section 3.2 for details). The migrations in the "threshold" strategy are triggered whenever the current serving MEH starts to have a positive carbon footprint, according to equation (3.12). Note that, for a fair comparison, the approaches we compare our strategy with are all based on the local resource allocation algorithm we devise in this chapter. Hence, their differing performance only depends on the adopted migration policy.

3.7.1 EASE performance varying the resources prediction window

EASE is evaluated by varying the local optimization window size T of MPC. By increasing it the controller is likely to find a better solution for the local management of resources and better estimates, which can be used in the migration process of phase 2. Fig. 3.3 shows the results of the aforementioned metrics for $T \in \{2, 5, 20\}$ time slots. Specifically, in Fig. 3.3a the processing power is shown as a function of the job generation probability p. While the curves for $T = 5$ and $T = 20$ substantially overlap, there is a slight increase in the energy consumption using $T = 2$

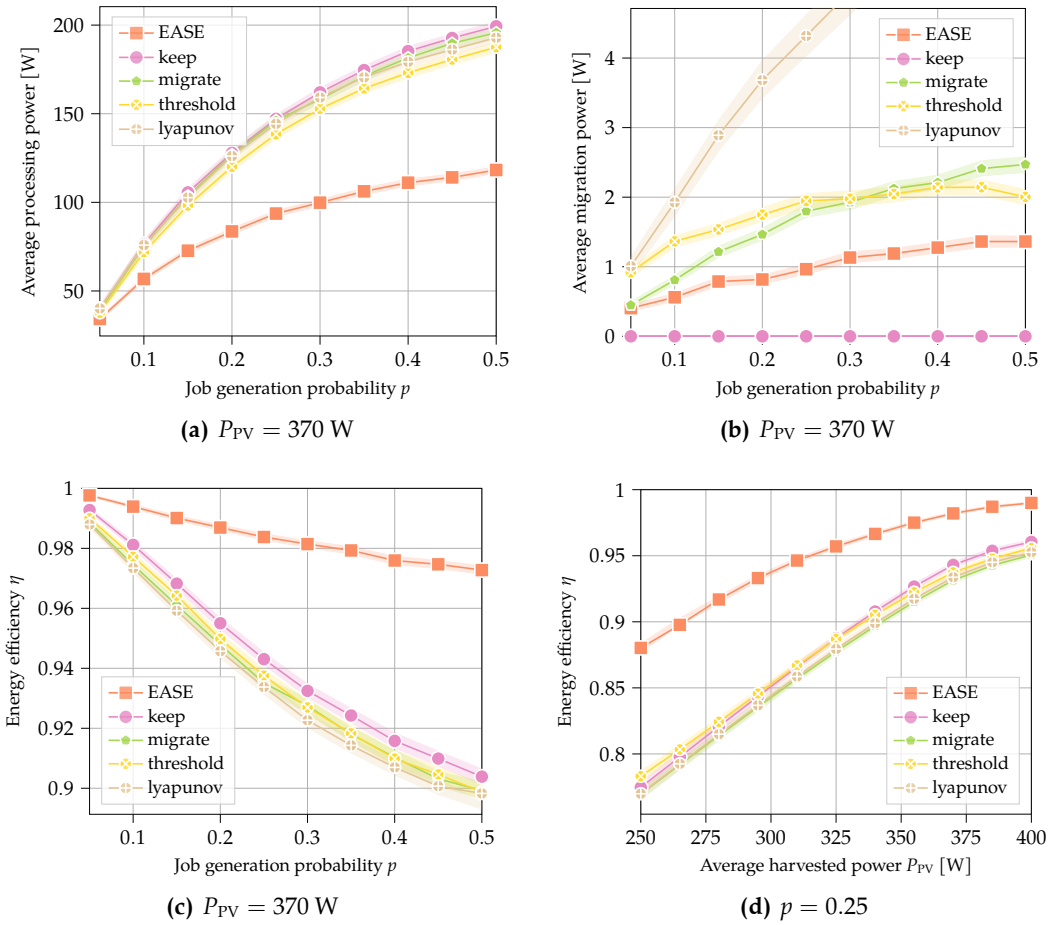


(a) $P_{PV} = 370$ W **(b)** $P_{PV} = 370$ W

(c) $P_{PV} = 370$ W **(d)** $p = 0.25$

Figure 3.3: Results of using EASE with different prediction windows for the local phase. Average processing (3.3a) and migration (3.3b) power dissipation of the edge servers. Energy efficiency with respect to the generation probability (3.3c) and the power generated by the PV cells (3.3d).

(of about 5%). For the migration power (Fig. 3.3b), the configuration that drains more energy is still $T = 2$, due to a poor prediction of future resources. However, setting $T = 5$ leads to a better migration efficiency than $T = 20$, but in the latter case the algorithm better captures the future system evolution, thus migrating the jobs to the next serving eNB at a slightly higher rate (see Table 3.6). The overall energy efficiency is depicted in Figs. 3.3c and 3.3d, showing that EASE is resilient to the prediction window size, as $T = 2$ loses at most 0.5% in efficiency when compared to the other two policies. In what follows, we select $T = 5$ to be compared with other existing strategies, as it provides the best tradeoff between performance and complexity.



(a) $P_{PV} = 370$ W **(b)** $P_{PV} = 370$ W

(c) $P_{PV} = 370$ W **(d)** $p = 0.25$

Figure 3.4: Comparison between EASE (T = 5) and other approaches of the literature. Average processing (3.4a) and migration (3.4b) power dissipation of the edge servers. Energy efficiency with respect to the generation probability (3.4c) and the power generated by the PV cells (3.4d).

3.7.2 EASE vs other migration methods from the literature

Fig. 3.4a shows the processing power, which has an increasing concave trend for all the strategies. As it can be seen, EASE allows substantial savings, e.g., as much as 70 W at $p = 0.5$ (a gain of 33%) with respect to the benchmarks. The "threshold" policy provides a slight improvement over the other heuristics, due to a better organization of the computational resources, as its migration decisions depend on energy considerations. The average power used to migrate the jobs is shown in Fig. 3.4b. Since the "keep" strategy never migrates tasks, its job migration power is always zero. On the other hand, the strategy with the highest migration power is "lyapunov", as it potentially migrates multiple replicas of the service to increase the probability of correctly following the user. The "migrate" and "threshold" strategies consume consistently more than the optimized EASE, as they blindly migrate services, even when the target MEH processes them inefficiently. In Fig. 3.4c, the energy efficiency η is shown as a function of the job generation probability. All the strategies

	EASE ($T = 2$)	EASE ($T = 5$)	EASE ($T = 20$)	keep	migrate	threshold	lyapunov
minimum latency jobs	33%	28%	30%	–	75%	58%	78%
drop rate	–	–	–	1.5%	0.5%	0.5%	1.5%

Table 3.6: Minimum latency executions and drop rates for $p = 0.3$ and $P_{\text{PV}} = 370$ W.

show an almost linear decrease for increasing p. However, the absolute slope of such a decrease is larger for the benchmark strategies with respect to EASE. At $p = 0.5$, EASE allows gaining about 7% in efficiency: the harvested energy can fully support the edge network for at least 97% of the total energy requirement. The energy efficiency is also evaluated by varying the amount of harvested energy (Fig. 3.3d), with the PV panel generating power in $[P^{\text{PV}}_{\text{min}}, P^{\text{PV}}_{\text{max}}]$ W. EASE can entirely sustain the edge at least 87.5% of the time when the harvested energy is at its minimum, i.e., $P_{\text{PV}} = 250$ W, leading to a gain of 10% concerning the other strategies, thus resulting in a significantly reduced carbon footprint. At $P_{\text{PV}} = 400$ W the gain is lower, but EASE performs very close to complete carbon neutrality (efficiency $\approx 99\%$). Note that 400 W are just sufficient to self-sustain (on average) the less powerful HP ProLiant server, but not the Nettrix computing unit at full load. As a final consideration, from Figs. 3.4c and 3.4d, it can be seen that the largest gain is achieved when either the computing demand is high (large p) or the harvested energy is scarce. These are the cases where it is important to use the available resources wisely, and EASE succeeds to do so.

The results about the jobs drop rate and the fraction of jobs finishing in the MEH co-located with the serving eNB (dubbed "minimum latency") are summarized in Tab. 3.6. In addition to being consistently more energy efficient, EASE never discards jobs, while the benchmark strategies drop a significant percentage of the tasks. The "migrate" and "lyapunov" strategies are the best in following the vehicles' trajectories, i.e., they seek to minimize the latency by transferring the jobs to the closest MEH. EASE takes a different approach, by considering latency deadlines, and seeking to migrate the jobs in a way that minimizes the overall energy that is drained from the power grid, subject to such deadlines. This leads to migration paths where jobs do not necessarily (strictly) follow the users. As a second-order optimization criterion, and only if feasible, EASE migrates jobs to the next predicted user location (eNB).

3.7.3 Rounding algorithm performance

To test the performance of the rounding Algorithm 3.3, the cost function (3.19) is evaluated with the obtained rounded solution $\boldsymbol{o}^r = \{\boldsymbol{o}^r_i \mid i \in \mathcal{N}\}$. The comparison is performed with the solution given by each server i simply following the desired $\bar{\boldsymbol{w}}_i$, i.e., the solution corresponding to the "migrate" strategy. Specifically, the ratio between the cost values of the "migrate" strategy and the rounded solution is computed, considering the cases where it is energetically inefficient to follow the desired migrations. Indeed, in the other case $\boldsymbol{o}_i = \boldsymbol{o}^r_i = \bar{\boldsymbol{w}}_i$, for all servers, i.e., $\bar{\boldsymbol{w}}_i$ is the optimal solution and it is a feasible one in the discrete domain, thus the costs are equal. As an example, with prediction horizon $T = 5$, job generation probability $p = 0.3$, and $P_{\text{PV}} = 300$ W, the gain of using the proposed relax and round optimization procedure of EASE over the "migrate"

strategy is on average 10 folds. More in the detail, the gain has a median of 3.8, the 10th percentile is 1.3, meaning that rarely a gain lower than 30% is observed, and the 90th percentile is 17. Hence, often, the rounding step of EASE induces a high gain over the blind "migrate" strategy from an energy perspective.

3.8 Conclusion

In this paper, we proposed EASE, a novel strategy for online job scheduling in a MEC-enabled network co-powered by the grid and renewable energy resources, considering an IoV scenario. EASE tackles the problem of ensuring computing service continuity as the users move within the resources-constrained network area. It allows deciding whether to migrate the jobs following the UE, or to continue the execution on the MEC server where it started. This is achieved through the alternation of a local control optimization phase, to estimate future resources, and a distributed consensus step, to reach the migration agreement. The primary objective is the minimization of the carbon footprint at the network side, guaranteeing adequate QoS to the moving users. Using EASE leads to energy efficiency improvements of up to 10% over heuristic strategies, getting close to carbon neutrality in a wide range of contexts.

4

Communication and computation overhead

This chapter is a comparison of the communication and computation efficiency of the approaches detailed in the previous two chapters. The reader might refer back to those chapters to recover the meaning of the parameters.

4.1 Convergence of the fully decentralized scheme of Chapter 2

In Fig. 4.1, numerical insights on the convergence time of the distributed MPC solution (Algorithm 2.3) are given, for a network with $M = 16$ servers, $N = 8$, and $\varepsilon = 0.5$. We observe that, while the optimization value does depend on the prediction error, i.e., larger errors lead to worse performance, the ADMM convergence speed is unaffected by this. Also, the term "iterations" is used as a proxy for the number of messages that are exchanged among the nodes, as a single local message is sent by each node at each ADMM iteration. The numerical solver uses CVXPY [103] and OSQP [104] 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. 4.1 is non-monotonic, and convergence is slower.

This is also confirmed by Fig. 4.2, 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 $\varepsilon \approx 0.6$, 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 the solution is highly sensitive to initial conditions, and may require

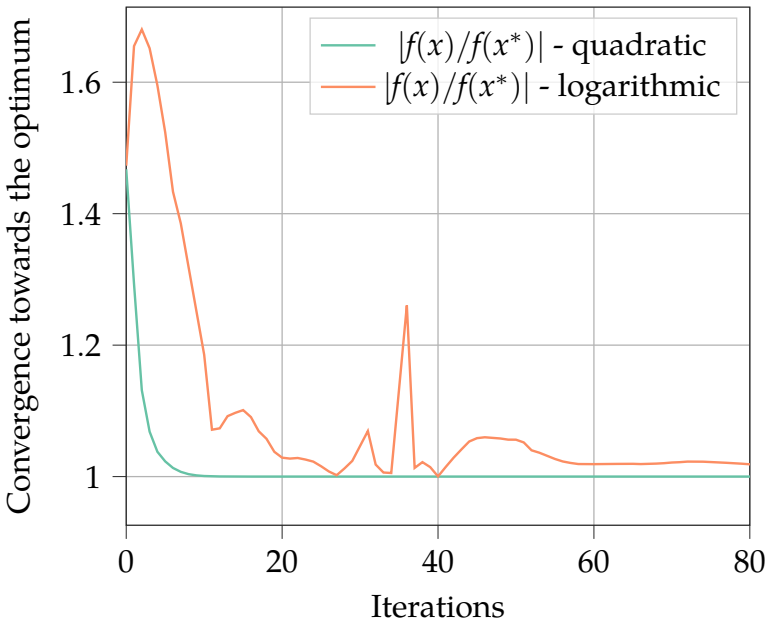


Figure 4.1: DRS convergence to the optimal value for $\varepsilon = 0.5$.

a higher number of iterations to converge. At very high loads (beyond $\varepsilon = 0.8$), the 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 $\varepsilon \geq 0.5$, where nearly all the servers are to be used anyway and, in turn, the convex cost represents a better choice. Using the logarithmic cost makes sense at a 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., $\varepsilon \approx 0.5$, 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 only leads to slower convergence, leading to the same solution attained by the quadratic cost formulation.

4.2 Convergence of the two steps dual ascent approach of Chapter 3

In Fig. 4.3, the convergence speed of the proposed decentralized solution is evaluated. Specifically, the cost value reached at the current iteration is compared with the optimal solution obtained with CVXPY [103], considering the absolute value of their ratio $|\Gamma(\boldsymbol{x}^+)/\Gamma(\boldsymbol{x}^*)|$. In the plot, the 90th percentile is shown, discarding hence 10% of outliers. Thus, whenever the ratio settles down to approximately 1, the nodes have reached the global minimum of the cost function. The results

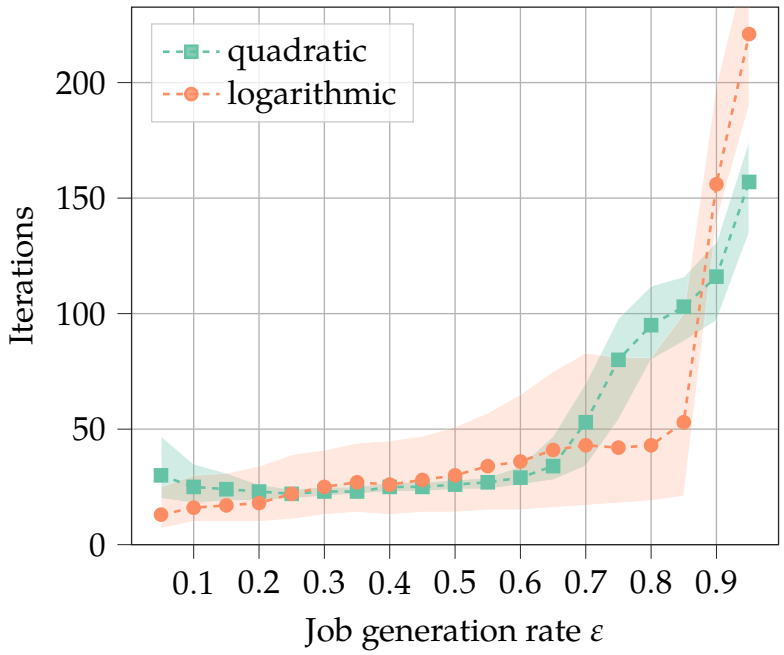


Figure 4.2: Median number of iterations of DRS to reach convergence for $z_{\text{obj}} = f_{\text{obj}} = 10^{-2}$. The shaded regions represent the interquartile range. The prediction window is set to $N = 8$.

show that the power availability impacts the convergence speed: the more harvested energy P_{PV} is available, the quicker the algorithm reaches the minimum. This descends from the fact that a high energy availability leads to a rare activation of the max term in function (3.17). When the max term returns 0 and the constraint (3.18) is not active, the optimum is simply given by $\boldsymbol{o}_i = \bar{\boldsymbol{w}}_i$, i.e., the selected action is to follow the vehicle movements. The nodes will be very fast in retrieving this particular solution, as the Lagrange multipliers associated with all the constraints remain null after the first two iterations, leading to accepting the solution. Similar reasoning holds for the job generation probability that determines the load of the servers. Specifically, the convergence requires more iterations as p increases. In fact, an increase in the average load experienced by the servers activates the constraint (3.18), modifying the optimal solution or even activating the penalties $\hat{\delta}_i$. As it is known, the dual ascent is slow when being close to constraint boundaries. However, as a general result, the number of iterations required to converge even with complex initializations is between 200 and 500. The communication overhead can be evaluated considering that two communication rounds (of a few bytes) are required per iteration (see Algorithm 3.2). Although this may actually appear to be a high number of exchanged messages, we remark that: i) the subsequent step of the proposed pipeline rounds the solution, and, in turn, it is not necessary to retrieve the exact optimum, but it is sufficient to obtain a decent cost value in the continuous domain; ii) we considered slots of $\tau = 3$ s, which is the amount of time available to make a migration decision. Longer time slots can be used, leaving more time for the decision process.

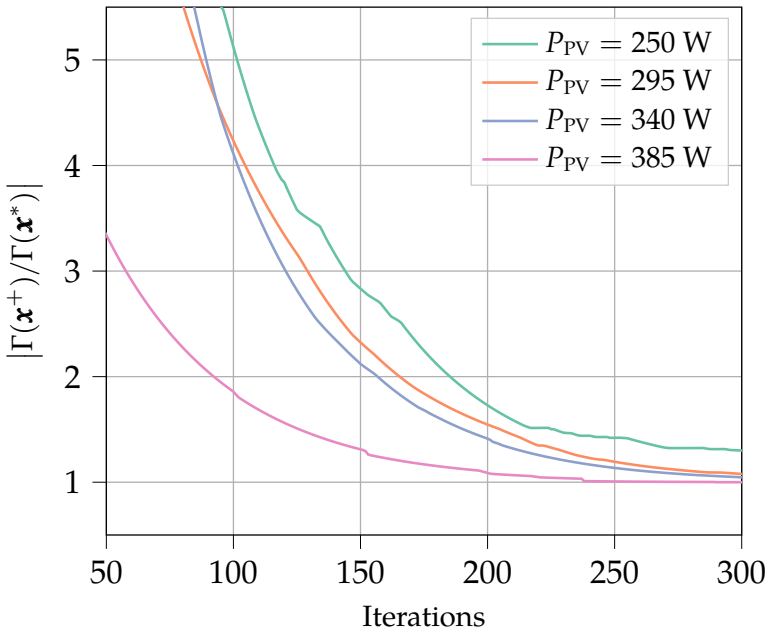


Figure 4.3: Ratio between the value of the cost at iteration m and the optimal cost computed with CVXPY (90th percentile). Job generation probability $p = 0.25$.

4.3 Complexity comparison

The decomposition approach adopted in EASE (Chapter 3) makes the overall algorithm feasible and lightweight to be run even in a complex and highly variable scenario such as the vehicular one considered. First, the nodes control the local processing and estimate the resource availability, then they agree on the offloading variables only. The work presented in Chapter 2, instead, uses MPC to jointly obtain an optimal decision on both the amount of workload to process locally and to offload to other MEHs in a fully decentralized fashion and considering the whole prediction window. This amounts to having a number of shared variables to be optimized via message passing $\mathcal{O}(VT)$, where V is the number of edges in the network graph and T is the prediction window. EASE, instead, by performing the preliminary local optimization phase, estimates the future *on average*, having thus a number of shared variables $\mathcal{O}(V)$. The local phase amounts to solving a constrained convex problem numerically every τ seconds, while the distributed phase requires broadcasting to the neighborhood (a part of) the primal and dual information of problem (3.20). In both cases, communication requires broadcasting a few bits only, which can be easily piggybacked on control packets that the MEHs normally exchange for other reasons. Although the amount of data exchanged within the single iteration is reduced by a factor $1/T$ in the case of EASE, the use of an accelerated version of the ADMM (the DRS with Anderson-II acceleration) of the solution proposed in Chapter 2 makes the number of iteration be an order of magnitude less than EASE: a few tens are required for the DRS approach while a few hundreds for the dual ascent. This is due to the better spectral properties of the ADMM algorithm compared to the dual

	communication rounds	messages per round	variables	local subproblems
DRS (Chapter 2)	$\sim 10^1$	1	$\mathcal{O}(VT)$	constrained convex
EASE (Chapter 3)	$\sim 10^2$	2	$\mathcal{O}(V)$	closed-form

Table 4.1: Summary of the communication and computation performance of the proposed distributed algorithms.

ascent. On the other hand, the solution proposed with EASE allows for a nice closed-form solution at every communication round even if the problem is non-differentiable. The solution proposed in Chapter 2 requires instead a QP solver at each communication round to solve a constrained convex subproblem, resulting in a higher computational burden.

4.4 Conclusion

A study on the communication and computation efficiency of the two pipelines proposed in the former chapters was presented here. In Table 4.1 a summary of the comparison is reported. In extreme synthesis, EASE requires more communication rounds (and two messages per round) to retrieve the optimal solution of the offloading problem but it is significantly more lightweight from a computational perspective, as it needs only inexpensive local closed-form updates. On the other hand, the approach based on the DRS, despite having better spectral and convergence properties, requires solving numerically a constrained convex subproblem at every iteration.

Overall, it is not clear yet which approach is the best one: it might likely depend on the medium access control protocols used for the messages exchange and on the computational power allocated for solving the subproblems at each iteration.

5
Concluding remarks

In this thesis, the problem of the management and operation of the MEC platform has been analyzed. A special focus was given to the integration in the infrastructure of RERs coming from PV panels, which can be easily equipped to eNBs. The general goal of the model-based optimization problems formulated was the minimization of the carbon footprint of the heterogeneous network infrastructure, considering both transmissions among edge servers and the execution of general-purpose jobs. In Chapter 2 the management of a hierarchical MEC network with MEHs equipped with EH devices and batteries is studied. Notably, the electricity trade with the power grid was also studied, as well as the use of different and simple resources and workload predictors. In Chapter 3 a vehicular scenario was considered, making the problem more complex with the additional requirement of following (if possible) the vehicles' trajectories when performing jobs migrations.

Given the decentralized nature of the edge infrastructure, algorithms based on message passing that only need to exchange information with the immediate neighborhood have been devised. Thus, an omniscient central orchestrator (network controller), which is often required in the works found in the literature, is not needed in what is presented here. Chapter 4 provides a study on the communication and computational burden of the distributed algorithms devised in this thesis.

The results compare the proposed solutions with simpler heuristics and other approaches found in the literature. They show that the predictive model-based algorithms devised outperform the benchmark in terms of the main optimization goal, i.e., the energy efficiency of the network edge, while also meeting the QoS requirements, formulated in terms of latency constraints. Notably, thanks to MPC, this is true even when the incoming availability of energy resources and computational load is not known with certainty but only on average if the optimization window is sufficiently large. Finally, there is a wide number of cases where the proposed solutions make the system almost completely carbon neutral, i.e., they use only the RERs.

5.1 Open challenges and future research directions

In light of the studies carried out during this doctorate, future research directions have also been identified. They are reported in the following list.

- **Application oriented algorithms:** scientific literature often either focuses on the latency or energy optimization of the MEC platform, as they are contrasting objectives. Nonetheless, an efficient infrastructure must respond both to the environmental challenges that the world is today facing *and* to the QoS requirements of the use cases of the future. Extreme low latency is fundamental for AR, healthcare monitoring, the tactile Internet, connected cars and remotely controlled unmanned vehicles, but also more playful applications such as online gaming. It is thus important to design solutions targeted for specific verticals to fully consider all the related requirements.

- **Energy profiling of edge devices:** models used to predict the energy consumption of CPUs and GPUs are outdated. The recent advances in the energy efficiency of the edge hardware produced by leading companies (AMD, Intel, and Nvidia to name a few) make it necessary to characterize the energy consumption of modern architectures concerning the computational load and the frequency, which is usually adaptive.

- **Testbed implementations:** too often in academia researchers resort to simulations. This is a required step but only the first one. On-field experimentation makes practical problems that engineers should consider arise. Proofs-of-concept should be thus realized with small testbeds to validate the operation of the designed algorithms for specific verticals.

- **Distributed learning:** the increasing use of machine and deep learning together with the massive availability of data coming from sensors (and users) make the edge computing facilities a suitable platform to train models. Federated learning (also known as local SGD) has developed recently as the candidate algorithm to train ML models distributedly. However, the high system heterogeneity of edge devices, i.e., from a computational and communication capacities perspective, makes this task challenging. Specific algorithms to tune the computation-communication tradeoff in heterogeneous systems need to be devised, taking into account i) the model accuracy, ii) the convergence time, and iii) the energy spent to train the model, as backpropagation operations are extremely expensive.

- **Neuromorphic computing:** recently introduced brain-inspired neural networks have the possibility of distributed and asynchronous firing [105]. Research in the field of microelectronics shows how this paradigm can reduce significantly both the time and energy used for processing information on standard hardware. It is thought that dedicated hardware will improve this result by orders of magnitude. Neuromorphic edge computing might be the solution in terms of both latency and energy for the network infrastructure of the future.

References

[1] A. Filali, A. Abouaomar, S. Cherkaoui, A. Kobbane, and M. Guizani, "Multi-access edge computing: A survey," *IEEE Access*, vol. 8, pp. 197 017–197 046, 2020.

[2] Q.-V. Pham, F. Fang, V. N. Ha, M. J. Piran, M. Le, L. B. Le, W.-J. Hwang, and Z. Ding, "A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art," *IEEE Access*, vol. 8, pp. 116 974–117 017, 2020.

[3] Cisco. (2020). Cisco annual internet report 2018-2023, [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[4] "State of the edge – a market and ecosystem report on edge computing," Equinix, Tech. Rep., 2021.

[5] C. Preist, D. Schien, and P. Shabajee, "Evaluating sustainable interaction design of digital services: The case of youtube," in *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019, pp. 1–12.

[6] The Shift Project. (2019). Lean ICT: towards digital sobriety, [Online]. Available: https://theshiftproject.org/wp-content/uploads/2019/03/Lean-ICT-Report_The-Shift-Project_2019.pdf.

[7] W. S. Jevons, *The coal question; an inquiry concerning the progress of the nation and the probable exhaustion of our coal-mines*. McMillan and Co., 1866.

[8] United Nations Organization (UNO). (2015). Transforming our world: the 2030 Agenda for Sustainable Development, [Online]. Available: https://sdgs.un.org/2030agenda.

[9] G. Perin, M. Berno, T. Erseghe, and M. Rossi, "Towards sustainable edge computing through renewable energy resources and online, distributed and predictive scheduling," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 306–321, 2022.

[10] B. Martinez and X. Vilajosana, "Exploiting the solar energy surplus for edge computing," *IEEE Transactions on Sustainable Computing*, vol. 7, no. 1, pp. 135–143, 2022.

[11] N. Shalavi, G. Perin, A. Zanella, and M. Rossi, "Energy efficient deployment and orchestration of computing resources at the network edge: A survey on algorithms, trends and open challenges," *ACM Computing Surveys*, (submitted), https://arxiv.org/pdf/2209.14141.pdf.

[12] J. Bi, H. Yuan, S. Duanmu, M. Zhou, and A. Abusorrah, "Energy-Optimized Partial Computation Offloading in Mobile-Edge Computing With Genetic Simulated-Annealing-Based Particle Swarm Optimization," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3774–3785, 2021.

[13] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, "Delay-Aware and Energy-Efficient Computation Offloading in Mobile-Edge Computing Using Deep Reinforcement Learning," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 3, pp. 881–892, 2021.

[14] N. Piovesan, M. Miozzo, and P. Dini, "Modeling the Environment in Deep Reinforcement Learning: The Case of Energy Harvesting Base Stations," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 8996–9000.

[15] D. Cecchinato, T. Erseghe, and M. Rossi, "Elastic and predictive allocation of computing tasks in energy harvesting iot edge networks," *IEEE Transactions on Network Science and Engineering*, 2021.

[16] Z. Wei, B. Zhao, J. Su, and X. Lu, "Dynamic Edge Computation Offloading for Internet of Things With Energy Harvesting: A Learning Method," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4436–4447, 2019.

[17] M. S. Munir, S. F. Abedin, N. H. Tran, Z. Han, E.-N. Huh, and C. S. Hong, "Risk-Aware Energy Scheduling for Edge Computing With Microgrid: A Multi-Agent Deep Reinforcement Learning Approach," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3476–3497, 2021.

[18] I. Labriji, F. Meneghello, D. Cecchinato, S. Sesia, E. Perraud, E. C. Strinati, and M. Rossi, "Mobility aware and dynamic migration of MEC services for the Internet of vehicles," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 570–584, 2021.

[19] G. Perin, F. Meneghello, R. Carli, L. Schenato, and M. Rossi, "EASE: Energy-aware job scheduling for vehicular edge networks with renewable energy resources," *IEEE Transactions on Green Communications and Networking*, 2022, (early access).

[20] D. Callegaro and M. Levorato, "Optimal Edge Computing for Infrastructure-Assisted UAV Systems," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 2, pp. 1782–1792, 2021.

[21] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016. eprint: `https://arxiv.org/abs/1602.05629`.

[22] E. Ozfatura, D. Gündüz, and H. V. Poor, "Collaborative Learning over Wireless Networks: An Introductory Overview," in *Machine Learning and Wireless Communications*, D. G. Y. Eldar A. Goldsmith and H. V. Poor, Eds., Cambridge University Press, 2022.

[23] Y. Sun, S. Zhou, Z. Niu, and D. Gündüz, "Dynamic Scheduling for Over-the-Air Federated Edge Learning With Energy Constraints," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 1, pp. 227–242, 2022.

[24] N. H. Tran, W. Bao, A. Zomaya, M. N. H. Nguyen, and C. S. Hong, "Federated Learning over Wireless Networks: Optimization Model Design and Analysis," in *IEEE Conference on Computer Communications (INFOCOM)*, Paris, France: IEEE, 2019, pp. 1387–1395.

[25] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, "Energy Efficient Federated Learning Over Wireless Communication Networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1935–1949, 2021.

[26] H. Xiao, J. Zhao, Q. Pei, J. Feng, L. Liu, and W. Shi, "Vehicle Selection and Resource Optimization for Federated Learning in Vehicular Edge Computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 1–15, 2021.

[27] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive Federated Learning in Resource Constrained Edge Computing Systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[28] B. Luo, X. Li, S. Wang, J. Huang, and L. Tassiulas, "Cost-Effective Federated Learning in Mobile Edge Networks," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3606–3621, 2021.

[29] L. Li, D. Shi, R. Hou, H. Li, M. Pan, and Z. Han, "To talk or to work: Flexible communication compression for energy efficient federated learning over heterogeneous mobile edge devices," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, Vancouver, BC, Canada: IEEE, 2021, pp. 1–10.

[30] Y. G. Kim and C.-J. Wu, "AutoFL: Enabling Heterogeneity-Aware Energy Efficient Federated Learning," in *MICRO-54: Annual IEEE/ACM International Symposium on Microarchitecture*, Virtual Event, Greece: IEEE Computer Society, 2021, pp. 183–198.

[31] J.-J. Kuo, C.-W. Ching, H.-S. Huang, and Y.-C. Liu, "Energy-Efficient Topology Construction via Power Allocation for Decentralized Learning via Smart Devices With Edge Computing," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 4, pp. 1806–1819, 2021.

[32] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[33] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Prentice hall Englewood Cliffs, NJ, 1989, vol. 23.

[34] 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`.

[35] 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, pp. 2322–2358, 2017.

[36] 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, pp. 94–101, 2018.

[37] J. Ni and X. Bai, "A review of air conditioning energy performance in data centers," *Renewable and sustainable energy reviews*, vol. 67, pp. 625–640, 2017.

[38] V. Chamola and B. Sikdar, "Solar powered cellular base stations: Current scenario, issues and proposed solutions," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 108–114, 2016.

[39] LG Electronics. (2016). LG Uplus demos solar-powered LTE base station in Daegwallyeong, [Online]. Available: `https://www.commsupdate.com/articles/2016/06/27/lg-uplus-demos-solar-powered-lte-base-station-in-daegwallyeong/`.

[40] C. Wu, "How to derive the maximum benefit from solar energy by using iiot-enabled energy storage systems," Moxa, Tech. Rep., 2017.

[41] R. Brännvall, M. Siltala, J. Gustafsson, J. Sarkinen, M. Vesterlund, and J. Summers, "Edge: Microgrid data center with mixed energy storage," in *Proceedings of the Eleventh ACM International Conference on Future Energy Systems*, 2020, pp. 466–473.

[42] Y.-J. Ku, S. Sapra, S. Baidya, and S. Dey, "State of energy prediction in renewable energy-driven mobile edge computing using cnn-lstm networks," in *2020 IEEE Green Energy and Smart Systems Conference (IGESSC)*, IEEE, 2020, pp. 1–7.

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

[44] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[45] 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, pp. 637–646, 2016.

[46] 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, pp. 1657–1681, 2017.

[47] 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`.

[48] 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, pp. 361–373, 2017.

[49] M. Berno, J. J. Alcaraz, and M. Rossi, "On the allocation of computing tasks under qos constraints in hierarchical mec architectures," in *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, IEEE, 2019, pp. 37–44.

[50] L. A. Barroso and U. Hölzle, "The Case for Energy-Proportional Computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[51] R. Sen and D. A. Wood, "Energy-Proportional Computing: A New Definition," *Computer*, vol. 50, no. 8, pp. 26–33, 2017.

[52] 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, pp. 15–26.

[53] 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, pp. 398–401, 2017.

[54] 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, pp. 3590–3605, 2016.

[55] A. Bozorgchenani, S. Disabato, D. Tarchi, and M. Roveri, "An energy harvesting solution for computation offloading in Fog Computing networks," *Computer Communications*, vol. 160, pp. 577–587, 2020.

[56] 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, pp. 79 877–79 886, 2019.

[57] 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)*, IEEE, 2018, pp. 1–6.

[58] 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 *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2020, pp. 8971–8975.

[59] M. Gaggero and L. Caviglione, "Predictive Control for Energy-Aware Consolidation in Cloud Datacenters," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 2, pp. 461–474, 2016.

[60] 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*, IEEE, 2015, pp. 81–90.

[61] 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)*, IEEE, 2017, pp. 1–6.

[62] P. Skarin, W. Tärneberg, K.-E. Årzen, and M. Kihl, "Towards Mission-Critical Control at the Edge and Over 5G," in *2018 IEEE International Conference on Edge Computing (EDGE)*, IEEE, 2018, pp. 50–57.

[63] 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, pp. 532–544, 2016.

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

[65] A. Makhdoumi and A. Ozdaglar, "Convergence Rate of Distributed ADMM Over Networks," *IEEE Transactions on Automatic Control*, vol. 62, no. 10, pp. 5082–5095, 2017.

[66] 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, pp. 2004–2009, 2015.

[67] 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, pp. 723–744, 2018.

[68] 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, pp. 756–766, 2011.

[69] 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, pp. 115–131.

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

[71] *Mobile edge computing (MEC); framework and reference architecture*, GS MEC 003 V2.2.1, ETSI, Dec. 2020.

[72] *Multi-access edge computing (MEC); MEC 5G integration*, GS MEC 031 V2.1.1, ETSI, Oct. 2020.

[73] J. Feng, Z. Liu, C. Wu, and Y. Ji, "Mobile edge computing for the Internet of vehicles: Offloading framework and job scheduling," *IEEE Vehicular Technology Magazine*, vol. 14, no. 1, pp. 28–36, 2019.

[74] *Multi-access edge computing (MEC); V2X information service API*, GS MEC 030 V2.1.1, ETSI, Apr. 2020.

[75] *Multi-access edge computing (MEC); study on mec support for V2X use cases*, GS MEC 022 V2.1.1, ETSI, 2018.

[76] *Handover procedures*, TS 23.009 V12.0.0, 3GPP, Sep. 2014.

[77] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and M. Atiquzzaman, "KEIDS: Kubernetes-based energy and interference driven scheduler for industrial IoT in edge-cloud ecosystem," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4228–4237, 2020.

[78] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2019.

[79] M. Feng, M. Krunz, and W. Zhang, "Joint task partitioning and user association for latency minimization in mobile edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 8, pp. 8108–8121, 2021.

[80] Z. Rejiba, X. Masip-Bruin, and E. Marín-Tordera, "A survey on mobility-induced service migration in the fog, edge, and related computing paradigms," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 1–33, 2019.

[81] T. V. Doan, G. T. Nguyen, H. Salah, S. Pandi, M. Jarschel, R. Pries, and F. H. P. Fitzek, "Containers vs virtual machines: Choosing the right virtualization technology for mobile edge cloud," in *Proceedings of the IEEE 2nd 5G World Forum (5GWF)*, 2019, pp. 46–52.

[82] S. Ramanathan, K. Kondepu, M. Razo, M. Tacca, L. Valcarenghi, and A. Fumagalli, "Live migration of virtual machine and container based mobile core network components: A comprehensive study," *IEEE Access*, vol. 9, pp. 105 082–105 100, 2021.

[83] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 140–147, 2018.

[84] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 2020–2033, 2019.

[85] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *Proceedings of the 10th ACM International Conference on Distributed and Event-Based Systems*, 2016, pp. 258–269.

[86] P. A. Frangoudis and A. Ksentini, "Service migration versus service replication in multi-access edge computing," in *Proceedings of the 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, 2018, pp. 124–129.

[87] I. Farris, T. Taleb, H. Flinck, and A. Iera, "Providing ultra-short latency to user-centric 5G applications at the mobile network edge," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 4, pp. 1–13, 2018.

[88] Q. Yuan, J. Li, H. Zhou, T. Lin, G. Luo, and X. Shen, "A joint service migration and mobility optimization approach for vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 9041–9052, 2020.

[89] C. Campolo, A. Iera, A. Molinaro, and G. Ruggeri, "MEC support for 5G-V2X use cases through docker containers," in *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, 2019, pp. 1–6.

[90] A. Aissioui, A. Ksentini, A. M. Gueroui, and T. Taleb, "On enabling 5G automotive systems using follow me edge-cloud concept," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 6, pp. 5302–5316, 2018.

[91] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on Markov decision process," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1272–1288, 2019.

[92] A. Dalgkitsis, P.-V. Mekikis, A. Antonopoulos, and C. Verikoukis, "Data driven service orchestration for vehicular networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4100–4109, 2021.

[93] I. Farris, T. Taleb, M. Bagaa, and H. Flick, "Optimizing service replication for mobile delay-sensitive applications in 5G edge network," in *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 1–6.

[94] A. Rago, G. Piro, G. Boggia, and P. Dini, "Anticipatory allocation of communication and computational resources at the edge using spatio-temporal dynamics of mobile users," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4548–4562, 2021.

[95] J. Li, X. Shen, L. Chen, D. P. Van, J. Ou, L. Wosinska, and J. Chen, "Service migration in fog computing enabled cellular networks to support real-time vehicular communications," *IEEE Access*, vol. 7, pp. 13 704–13 714, 2019.

[96] E. Björnson, J. Hoydis, and L. Sanguinetti, "Massive MIMO networks: Spectral, energy, and hardware efficiency," *Found. Trends Signal Process.*, vol. 11, no. 3–4, pp. 164–655, 2017.

[97] J. Ko, Y. Cho, S. Hur, T. Kim, J. Park, A. F. Molisch, K. Haneda, M. Peter, D. Park, and D. Cho, "Millimeter-wave channel measurements and analysis for statistical spatial channel model in in-building and urban environments at 28 GHz," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5853–5868, 2017.

[98] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao, "Performance and energy modeling for live migration of virtual machines," in *Proceedings of the 20th international symposium on High performance distributed computing*, 2011, pp. 171–182.

[99] Standard Performance Evaluation Corporation. (2021). SPECpower results, [Online]. Available: `https://www.spec.org/power_ssj2008/results/`.

[100] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Athena Scientific, 1999.

[101] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3-4, pp. 128–138, 2012.

[102] SUMO. (2021). TAPAS Cologne scenario, [Online]. Available: `https://sumo.dlr.de/docs/Data/Scenarios/TAPASCologne.html`.

[103] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[104] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, pp. 1–36, 2020.

[105] D. Marković, A. Mizrahi, D. Querlioz, and J. Grollier, "Physics for neuromorphic computing," *Nature Reviews Physics*, vol. 2, no. 9, pp. 499–510, 2020.

List of publications

Journals

[9] G. Perin, M. Berno, T. Erseghe, and M. Rossi, "Towards sustainable edge computing through renewable energy resources and online, distributed and predictive scheduling," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 306–321, 2022.

[11] N. Shalavi, G. Perin, A. Zanella, and M. Rossi, "Energy efficient deployment and orchestration of computing resources at the network edge: A survey on algorithms, trends and open challenges," *ACM Computing Surveys*, (submitted), `https://arxiv.org/pdf/2209.14141.pdf`.

[19] G. Perin, F. Meneghello, R. Carli, L. Schenato, and M. Rossi, "EASE: Energy-aware job scheduling for vehicular edge networks with renewable energy resources," *IEEE Transactions on Green Communications and Networking*, 2022, (early access).

Conferences

[106] G. Perin, G. Fighera, and L. Badia, "Comparison of Nash Bargaining and myopic equilibrium for resources allocation in cloud computing," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[107] G. Perin, D. Nophut, L. Badia, and F. H. Fitzek, "Maximizing airtime efficiency for reliable broadcast streams in WMNs with multi-armed bandits," in *2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 2020, pp. 0472–0478.

[108] G. Perin, A. Buratto, N. M. Anselmi, S. Wagle, and L. Badia, "Adversarial jamming and catching games over AWGN channels with mobile players," in *2021 17th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2021, pp. 319–324.

[109] G. Perin and L. Badia, "Reinforcement learning for jamming games over AWGN channels with mobile players," in *2021 IEEE 26th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2021, pp. 1–6.

Book Chapter

[110] F. Meneghello, G. Perin, and M. Rossi, *"Machine Learning Techniques for Context Extraction and User Profiling in 5G Mobile Systems,"* book chapter in *"CNIT technical report 07- Machine Learning and 5G/6G Networks: Interplay and Synergies"*, S. Barbarossa and A. Zanella, Eds. Texmat, Jul. 2021.

Acknowledgments

I'm writing these lines on the afternoon of September 29th, right after finishing the abstract. Maybe for the first time in three years, I'm thinking serenely about my journey as a Ph.D. student. Looking back, I'm happy and sure I would do it again. Even though I made some mistakes, I don't regret them because they are partially responsible for making me the person I am today and I'm thankful to them. And I'm thankful to all the people that traveled with me and whom I will list here, although I'll certainly forget to name some.

First of all, I am thankful to my supervisor, Prof. Michele Rossi. Despite all of his academic commitments, he was able to give me a direction for my research activity, teach me how to write a scientific paper and a project proposal, and the patience required in the world of research when results don't want to collaborate, a natural gift which I don't have. I'm also thankful to all the Professors of the University of Padova who helped me and collaborated on my works: Tomaso Erseghe, Leonardo Badia, Andrea Zanella, Ruggero Carli, and Luca Schenato. A special mention goes to my colleagues of the MR's team: Marco Canil, Riccardo Mazzieri, Francesca Meneghello, Jacopo Pegoraro, and Silvia Zampato. They, together with Martina Capuzzo, have been for me a great group of friends other than colleagues. Thanks also to Laura Crosara, with whom I shared the task of tutoring the course of Telecommunications for the callow students of the Bachelor of Information Engineering. I will also thank all the other Ph.D. students: listing all of them would be too long.

I am thankful for the collaboration with INRIA Nice, with Dr. Giovanni Neglia and Othmane Marfoq, thanks to whom I approached federated learning, and for my period abroad at the University of California, Irvine, under the supervision of Prof. Marco Levorato. In the US I met many people that I will remember with pleasure, including my roommates Vida Rebello and Jason Yang, the colleagues Sharon Ladròn, Forough Shirin Abkenar, Ian Harshbarger, and Luis Florenzan and his roommates and Italian friends André Abtahi, Raquel Bowman, Silvia Perzolli, and Giulia Casalini. A special thanks also to my Russian friend Kirill Kurdin, Mark Strickland, Tony Alexos, and all the friends of St. Paul's Church. Finally, I want to mention Morgan Penning, who shared with me a large part of the summer, drove me to the airport the very last day, and gave a look at the introduction of my thesis sharing her knowledge of English.

I am thankful for volleyball and the choir. They have always been faithful friends when I needed to focus on something else. And to the male national team of volleyball: we are European and World champions after too many years of disappointments. This young group gave me a reason to smile even in harsh moments.

Although our paths parted, I want to thank Letizia, with whom I shared the majority of my life as a Ph.D. student. She always believed in my abilities as a researcher. Last but not least, I will thank my family and close friends: Alberto, Antonella, and Giulia, who recently got her Master's degree in Evolutionary Biology with honors, making us all proud. Quarrels are present

in everyday life but I know they love me. And thanks to my extended family: my grandparents Giuseppe and Maria Bruna, my uncles, aunts, and cousins Flavio, Doriana, Eleonora, Chiara and Alessandro, and Giorgio, Giulia, Anna, and Laura. Finally, a big thank you to my best friends Alex, the person I can always count on, Stefano, and also Martina, who is always trying to solve the hard task of finding me a girlfriend.