On the Allocation of Computing Tasks under QoS Constraints in Hierarchical MEC Architectures

Michele Berno*, Juan José Alcaraz†, Michele Rossi*
*Dept. of Information Engineering, University of Padova, Italy
†Dept. of Information and Communications Technologies, Technical University of Cartagena, Spain
email: michele.berno@dei.unipd.it, juan.alcaraz@upct.es, rossi@dei.unipd.it

***Abstract*—In this study, a model for the allocation of processing tasks in Mobile Edge Computing (MEC) environments is put forward, whereby a certain amount of workload, coming from the base stations at the network edge, has to be optimally distributed across the available servers. At first, this allocation problem is formulated as a centralized (offline) optimization program with delay constraints (deadlines), by keeping into account server qualities such as computation speed and cost, and by optimally distributing the workload across a hierarchy of computation servers. Afterwards, the offline problem is solved devising a *distributed* algorithm, utilizing the Alternating Direction Method of Multipliers (ADMM). Selected numerical results are presented to discuss the key features of our approach, which provides control over contrasting optimization objectives such as minimizing the energy consumption, balancing the workload, and controlling the number of servers that are involved in the computation.**

***Index Terms*—Mobile Edge Computing, distributed optimization, Alternating Direction Method of Multipliers.**

I. Introduction

Edge Computing's (EC) primary objective is to place computation capabilities and services at the network edge, closer to the mobile user. EC will enable cloud computing capabilities and network services at ultra-low latency. Furthermore, it will allow the exploitation, in near realtime, of local information about the radio access network conditions, while also reducing the load in the core network.

Since 2014, Mobile Edge Computing or Multi-access Edge computing (MEC), has been supported by the European Telecommunications Standards Institute (ETSI), whose aim is to create a standardized environment granting the interoperability of different MEC platforms and service providers. Among its use cases, ETSI reports video analytics, Augmented and Virtual Reality (AR/VR), Internet of Things (IoT), and optimized local content distribution. Many telecommunication companies are actively engaged in the EC domain, and are developing their own MEC solutions. Especially, MEC is expected to be useful in future IoT networks where, as estimated by Cisco, as many as 50 billion devices may be connected by 2020, and also in mobile networks, where MEC could be utilized to serve application needs (e.g., image processing), or to speed up and improve some of the computing tasks that are required to manage the mobile network itself [1].

In this paper, we consider a mobile network featuring a hierarchical system of MEC servers. Each Base Station (BS) has its own (co-located) MEC server, which is devoted to processing the work generated by the connected mobile users. Incoming requests are classified according to the type required processing intensity (CPU cycles per bit) and processing deadline. Besides, a number of additional servers is located in a second tier, with the role of relieving the BS servers from some of the computation burden. In this work, we focus on a two-layer hierarchical architecture, where computation jobs can be offloaded to the servers in either layer.

Specifically, we devise an optimization framework to perform optimal workload allocations among the servers in the first (co-located with the BSs) and in the second tier. This framework accounts for key objectives such as balancing the workload across the available servers, controlling the amount of cooperation (i.e., keeping the number of cooperating servers low) and minimizing the total energy drained by the network for the computing tasks. Centralized and distributed optimal algorithms are devised accounting for these *conflicting* goals, while ensuring that all processing deadlines are met. The distributed solution is obtained through the Alternating Direction Method of Multipliers (ADMM) [2]. While being amenable to several extensions, discussed in Section VII, our formulation makes it possible to effectively control non-trivial and conflicting goals, such as minimizing the energy consumption, balancing the workload across the computing servers and making the solution sparse (i.e., keeping the number of active servers low). Load balancing is here controlled via a parameter and expressed as a tunable *consensus* constraint. The smaller this parameter, the tighter the constraints and the more evenly distributed is the workload. In Section VI, selected numerical results are shown to demonstrate the effectiveness of the proposed algorithms.

The paper is structured as follows. A review of the existing literature is provided in Section II, the system model is presented in Section III, whereas the centralized and the distributed formulations are respectively detailed in Sections IV and V. Numerical results are described in Section VI and conclusions are drawn in Section VII.

II. State of The Art

MEC is a lively area of research, as testified by recent surveys on the topic [3]. It initially implied the use of BSs for offloading computation tasks from mobile terminals and has now a broader scope, encompassing any device with computing capabilities, including resource constrained constrained IoT nodes and autonomous vehicles. From the user terminal side, the main challenge amounts to deciding which computing tasks are to be offloaded, and how. From the edge side, the main challenges are mobility management and an efficient allocation of computing resources across the MEC servers, see, e.g., [4]. Our work focuses on the latter aspect, i.e., optimizing how processing is handled within the edge network.

There are three different ways in which tasks can be distributed within the MEC infrastructure. In a first one, the tasks coming from each BS are always offloaded to a predefined MEC node, remaining there until completion [5]. This approach, however, poorly copes with spatially imbalanced computation requests. To address this, a second approach allows MEC nodes to share their workloads. This concept has been referred to in the literature as MEC load balancing [6], MEC clustering [7], BS peer offloading [8], or simply cooperation [9]. The third approach contemplates the option of submitting tasks to cloud servers located at higher hierarchy levels in the computing infrastructure [10]. This is generally used in conjunction with MEC load balancing, as done in the present work.

The typical objectives pursued by MEC resource allocation algorithms are: minimizing power consumption, minimizing execution delay, or maximizing revenue. In some studies, these objectives are combined, e.g., through a weighted sum of power consumption and delay [11] or by solving two problems at different timescales (revenue and delay) [12]. Even in works where energy efficiency is not the main objective, power consumption is usually present. For example, by establishing energy constraints [8].

The degree of detail used to model offloaded tasks is also a distinctive feature of existing works. While some studies [10], [12] assume indistinguishable tasks (e.g., in terms of computing effort per transmitted bit), other works [13], [14] consider heterogeneous tasks. Although execution delay is almost always considered, only some works (e.g. [7], [15]) consider execution deadlines into the formulation.

Our main contributions are summarized as follows:

- We extend previous work through a more complete optimization framework, by jointly considering hard-processing deadlines, heterogeneous tasks' requirement (in terms of deadline and required computational resources), and optimizing over conflicting goals, namely, *i)* energy consumption, *ii)* workload balancing (controlled via a parameter θ and expressed as a tunable consensus constraint) and *iii)* number of active servers (promoting some sparsity in the server allocation). The priority of goals *i)* and *ii)* is controlled though two parameters γ_1 and γ_2. The resource allocation problem for such system is formulated as an offline centralized optimization problem.
- Then, we present a distributed algorithm to solve the centralized problem. The technique we have used, called Alternating Direction Method of Multipliers (ADMM), decomposes the whole problem into smaller sub-problems, one for each edge node. The partial solutions found by the edge nodes are then jointly steered toward a global (optimal) solution using an iterative procedure.
- Lastly, we report some preliminary results, obtained using a simulator written in Matlab, about the flexibility offered by our framework in accommodating multiple conflicting goals, and we show ADMM convergence rate for some parameter's choices.

III. System Model

We consider a Mobile Edge Computing (MEC) architecture comprising M_s computing entities or *servers*, organized in two tiers. The first tier contains $M_{s^{(1)}} < M_s$ servers, referred to as the *edge* servers, which are located at $M_{s^{(1)}}$ Base Stations (BSs). Further $M_{s^{(2)}} = M_s - M_{s^{(1)}}$ servers operate within a second tier. For later convenience, we define the set of indices $\mathcal{M}_{s^{(1)}} = \{1, \dots, M_{s^{(1)}}\}$ and $\mathcal{M}_s = \{1, \dots, M_s\}$. Each BS in set $\mathcal{M}_{s^{(1)}}$ receives computation offloading requests from the user terminals under its coverage area, and transfers these requests to its co-located edge server. An edge server $i \in \mathcal{M}_{s^{(1)}}$ receiving a computing request from its BS can either process it *locally* or transfer such request (or a portion of it) to any other server in $\mathcal{M}_s \setminus \{i\}$. Once a computing task is processed, the computation result is sent back to the user terminal that originated the computing request.

With $\mathcal{N}_i$ we denote the set of servers that are direct neighbors to server i, meaning that i and $j \in \mathcal{N}_i$ are connected via a *bi-directional* and *direct* communication channel. The nature of such channel (either wireless or wired) in this study is abstracted by the delay incurred in transmitting a task (and the corresponding result) over it. With $\mathcal{N}_i^*$, we define the set of neighbors of node i which belong to the first tier and that, in turn, can possibly offload some of the collected computation requests.

In this paper, we do not allow for multiple forwarding of the same workload. Accordingly, the offloading requests sent from a server need to be handled by the receiving server, since it cannot act as a relay. This amounts to the following processing models: the edge server co-located with a BS can 1) process the workload locally, 2) outsource some of its workload to a server located within the second tier, or 3) outsource some of its workload to another edge server, that is still located within the first tier.

Table I summarizes the notation used throughout the paper.

A. Workload gathered from the BS

We consider Q different types of computing tasks, each characterized by a *task specific* computation intensity I_q, expressed in CPU cycles per bit, and by a completion deadline τ_q, measured in seconds, for $q \in \mathcal{Q} = \{1, \dots, Q\}$. For each task type q, we define W_i^q as the average amount of workload, in bits, that is collected by the edge server $i \in \mathcal{M}_{s^{(1)}}$ (i.e., received from its co-located BS[1]). The total workload generated at server i is the sum of the workloads of each type, $W_i^1 + \dots + W_i^Q$.

B. Workload distribution and resource allocation

Each server i has a total computational power of F_i cycles per second. Servers can make two types of decisions: workload distribution decisions and decisions about the amount of local CPU share that is assigned to a given task, see Fig. 1.

[1]We assume that servers at the second tier do not generate workload, and only process the tasks transferred from the edge servers. However, this aspect can be relaxed without major changes in the model.

Notation	Description
$\mathcal{M}_s$	Set of processing servers ($\lvert\mathcal{M}_s\rvert = M_s$)
$\mathcal{M}_{s^{(h)}}$	Set of servers at tier h, ($h = 1$ edge servers) ($\lvert\mathcal{M}_{s^{(1)}}\rvert = M_{s^{(1)}}$)
$\mathcal{N}_i$	Set of neighbors of server i
$\mathcal{N}_i^*$	Set of 1^{st} tier neighbors of server i
$\mathcal{Q}$	Set of workload types ($\lvert\mathcal{Q}\rvert = Q$)
I_q	Computation intensity (CPU cycles per bit) for $q \in \mathcal{Q}$
τ_q	Completion deadline for type q tasks
W_i^q	Type q workload (in bits) collected at BS $i \in \mathcal{N}_s^{(1)}$
w_{ij}^q	Amount of type q workload transferred from server i to j
$\tilde{w}_{ij}^q$	Estimate of w_{ij}^q at server j
$\boldsymbol{w}_i$	Workload distribution decision vector at server i
α_{ij}^q	Fraction of CPU resources of server i assigned to w_{ji}^q
$\boldsymbol{\alpha}_i$	Vector of CPU shares assigned to tasks by server i
$\boldsymbol{x}_i$	Local decision variables regarding server i
$\boldsymbol{x}$	Global decision variables
F_i	Processing power (CPU cycles per second) of server i
d_{ij}	Round-trip time delay between i and j (in seconds)
c_i	Power consumption coefficient per CPU cycle for server i
$f_E^{(i)}$	Objective function encoding the energy consumption cost
$f_C^{(i)}$	Objective function encoding the cooperation cost
ϕ_i	Load factor for server i
$\boldsymbol{z}$	Global consensus variables
$\boldsymbol{\lambda}$	Dual variables

TABLE I
LIST OF SYMBOLS USED IN THE PAPER.

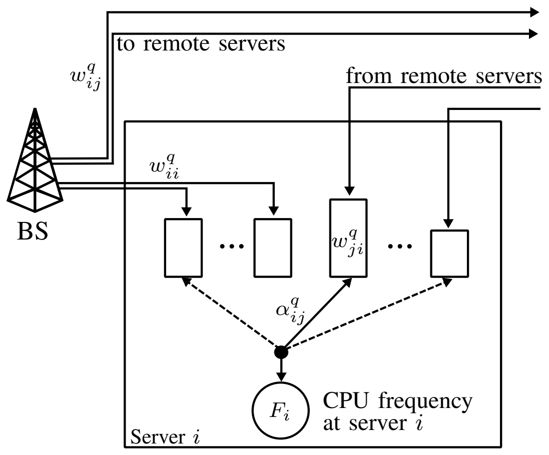


Fig. 1. MEC edge server $i \in M_{s^{(1)}}$: part of the workload (w_{ij}^q) is transferred to remote server j. Similarly, some workload may be received from remote server j (w_{ji}^q). The local variable $\alpha_{ij}^q \in [0,1]$ indicates the fraction of the CPU power at server i that is used to process workload w_{ji}^q.

The decision variables are collected into a global vector $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\mathcal{M}_s})$, where the component $\boldsymbol{x}_i = (\boldsymbol{w}_i, \boldsymbol{\alpha}_i, \phi_i)$ represent the local decision variables to server i, which will be detailed shortly.

1) Workload distribution decisions: w_{ij}^q denotes the amount of workload of type q that is transferred from server i to server j. Note that w_{ii}^q denotes the workload of type q gathered at server i and processed locally. The workload distribution decisions made by server i are collected into vector $\boldsymbol{w}_i = (w_{ij}^q)_{j \in \mathcal{N}_i \cup \{i\}, q \in \mathcal{Q}}$. A server can only transfer its own generated workload. Therefore, $\sum_{j \in \mathcal{N}_i} w_{ij}^q + w_{ii}^q = W_i^q$. This constraint is expressed as:

$$\boldsymbol{x}_i \in \mathcal{X}_1^{(i)} = \left\{ \boldsymbol{x}_i : \sum_{j \in \mathcal{N}_i} w_{ij}^q + w_{ii}^q = W_i^q, \ \forall q \in \mathcal{Q} \right\} . \quad (1)$$

For the whole network, this constraint amounts to $\boldsymbol{x} \in \mathcal{X}_1 = \mathcal{X}_1^{(1)} \times \mathcal{X}_1^{(2)} \times \cdots \times \mathcal{X}_1^{(\mathcal{M}_{s^{(1)}})}$. For the servers in the second tier we have $w_{ii}^q = 0$, and they do not generate computation requests.

2) Allocation of CPU shares: F_i is the total computational power (CPU cycles per second) of server i, and α_{ij}^q denotes the portion of F_i that server i allocates to the q-type workload it receives from server j. Note that $\sum_{q \in \mathcal{Q}} \sum_{j \in \mathcal{N}_i^* \cup \{i\}} \alpha_{ij}^q + \alpha_i^{\text{idle}} = 1, \ \forall i$, i.e., a server cannot allocate more than 100% of its computational resources.

The resource allocation decision vector at server i is denoted by $\boldsymbol{\alpha}_i = (\{\alpha_{ij}^q\}_{j,q}, \alpha_i^{\text{idle}})$ for $j \in \mathcal{N}_i^* \cup \{i\}, q \in \mathcal{Q}$. The constraint on CPU resources is:

$$\boldsymbol{x}_i \in \mathcal{X}_2^{(i)} = \left\{ \boldsymbol{x}_i : \sum_{q \in \mathcal{Q}} \sum_{j \in \mathcal{N}_i^* \cup \{i\}} \alpha_{ij}^q + \alpha_i^{\text{idle}} = 1 \right\} , \quad (2)$$

that network-wise become $\boldsymbol{x} \in \mathcal{X}_2 = \mathcal{X}_2^{(1)} \times \cdots \times \mathcal{X}_2^{\mathcal{M}_{s^{(1)}}}$.

C. Computation and delay

We follow the processing model described of [3], and widely used in previous papers [16], [17]. If edge server $i \in \mathcal{M}_{s^{(1)}}$ assigns some workload $w_{ij}^q > 0$, then server j will hold one queue for this workload, which requires a total of $I_q w_{ij}^q$ CPU cycles to be processed. Hence, in a fully connected topology, each server can hold up to $M_{s^{(1)}} Q$ parallel queues. If server i allocates a fraction α_{ij}^q of its computational resources to process w_{ji}^q from server j, the associated execution latency is given by $I_q w_{ji}^q / (\alpha_{ij}^q F_i)$. With d_{ji} we denote the average round-trip time[2] between servers j and i ($d_{ij} = d_{ji}$ for all i, j, and $d_{ii} = 0$). The completion time for a task generated at node j and transferred to node i is given by its average execution latency at i and the average round-trip time between j and i. If τ_q is the *completion deadline* associated with task type q, the following delay constraint holds for every $q \in \mathcal{Q}$:

$$\frac{I_q w_{ji}^q}{\alpha_{ij}^q F_i} + d_{ji} \leq \tau_q \ , \ \forall i \in \mathcal{M}_{s^{(1)}} \ , \ j \in \mathcal{N}_i^* \ . \quad (3)$$

Eq. (3) can be more conveniently rewritten by multiplying each side by $\alpha_{ij}^q F_i$. For each server $i \in \mathcal{M}_s$ we have:

$$\boldsymbol{x} \in \mathcal{X}_3 = \left\{ \boldsymbol{x} : \ I_q w_{ji}^q + F_i \alpha_{ij}^q (d_{ji} - \tau_q) \leq 0, \right.$$
$$\left. i \in \mathcal{M}_s, \ j \in \mathcal{M}_{s^{(i)}}, \ \ q \in \mathcal{Q} \right\} . \quad (4)$$

D. Load factor

Let $\phi_i \in [0, +\infty]$ be the load factor experienced by server i. For every server $i \in \mathcal{M}_s$, we define it as

$$\phi_i = \frac{1}{F_i} \sum_{j \in \mathcal{N}_i^* \cup \{i\}} \sum_{q \in \mathcal{Q}} \frac{I_q w_{ji}^q}{(\tau_q - d_{ji})} \, . \quad (5)$$

[2]Note that our model can be easily extended to the case where the delay depends on q, that is d_{ji}^q.

The rationale behind (5) is that $F_i \hat{\alpha}_{ij}^q = I_q w_{ji}^q/(\tau_q - d_{ji})$ represents the minimum number of CPU cycles per second that would be required to process workload w_{ji}^q within its deadline τ_q. Summing over all nodes j that can possibly send workload to server i and over all workload types q returns the total minimum number of cycles needed so that all tasks can be processed on time at server i. Dividing such sum by the CPU frequency at server i (F_i) may possibly lead to: 1) $\phi_i > 1$, the scheduled workload cannot be processed meeting all deadlines and the schedule is *infeasible*, 2) $\phi_i \leq 1$, the workload scheduled at server i is all processed on time, meeting all the delay constraints. Through the following set $\mathcal{X}_4$, we relate ϕ_i to the other system variables:

$$\boldsymbol{x} \in \mathcal{X}_4 = \left\{ \boldsymbol{x} : \phi_i = \frac{1}{F_i} \sum_{j \in \mathcal{N}_i^* \cup \{i\}} \sum_{q \in \mathcal{Q}} \frac{I_q w_{ji}^q}{(\tau_q - d_{ji})}, i \in \mathcal{M}_s \right\} \tag{6}$$

E. Control of load imbalance

We wish to balance the processing workload that is allocated across servers. In this paper, we do so by constraining ϕ_i, $i \in \mathcal{M}_s$, to fall within a maximum distance of $\theta \in [0, 1]$ from any other load factor ϕ_j:

$$\boldsymbol{x} \in \mathcal{X}_5 = \left\{ \boldsymbol{x} : \ -\theta \leq \phi_i - \phi_j \leq \theta, \quad \forall i, j \in \mathcal{M}_s \right\}. \tag{7}$$

The constant θ allows tuning how loose the matching of the load factor across the network's nodes can be. Small values of θ force homogeneous allocation of the workload across the servers. In the limit, $\theta = 0$, we solve a distributed *consensus* problem, where load factors must be all equal to one another.

F. Ensuring correctness in the allocated CPU shares

Next, we present an important technical constraint. What we would like to obtain is that any server i can assign a non-zero share of its own CPU resources, i.e., $\alpha_{ij}^q > 0$, only if there is an actual workload $w_{ji}^q > 0$ that is being transferred from server j to server i. The following constraint ensures this:

$$\boldsymbol{x} \in \mathcal{X}_6 = \left\{ \boldsymbol{x} : \ \alpha_{ij}^q \leq w_{ji}^q, \ i \in \mathcal{M}_s, \ j \in \mathcal{M}_{s^{(1)}}, \ q \in \mathcal{Q} \right\}. \tag{8}$$

IV. Problem Formulation

A. Optimization Objectives

The optimization problem considers the following (contrasting) goals: 1) controlling the network-wide energy consumption due to computing, 2) balancing the computing workload across servers, and 3) controlling the number of cooperating servers, promoting local over distributed processing. These criteria are further discussed next.

1) Minimization of the energy consumption: for a cost-effective operation of the MEC servers, a sensible optimization criterion corresponds to minimizing the energy expenditure associated with the computing tasks. The energy consumption of a CPU cycle at server i is denoted by c_i. Therefore, the energy consumption associated with the processing of workload w_{ji}^q is given by $c_i I_q w_{ji}^q$. The energy $f_i^E(\boldsymbol{x})$ spent at server $i \in \mathcal{M}_s$, and the total energy $f^E(\boldsymbol{x})$ spent across the whole network, are:

$$f^E(\boldsymbol{x}) = \sum_{i \in \mathcal{M}_s} f_i^E(\boldsymbol{x}) = \sum_{i \in \mathcal{M}_s} c_i \left(\sum_{j \in \mathcal{N}_i^*} \sum_{q \in \mathcal{Q}} I_q w_{ji}^q + I_q w_{ii}^q \right). \tag{9}$$

We observe that the energy spent by server i is a function of the workload received from the other servers and, in turn, the objective function is not separable in the local variables (vector $\boldsymbol{x}_i$). We shall discuss more on this in the next section.

2) Cooperation control: in some cases, being able to control the number of cooperating servers can be useful, and this in our scenario amounts to controlling the number of active edges (over which some non-zero workload is transferred). The cost functions that are utilized to this end, for a single server (f_i^c) and network-wide (f^c), are:

$$f^C(\boldsymbol{x}) = \sum_{i \in \mathcal{M}_s} f_i^C(\boldsymbol{x}_i) = \sum_{i \in \mathcal{M}_s} \sum_{j \in \mathcal{N}_i^*} \sum_{q \in \mathcal{Q}} \frac{|w_{ij}^q|}{F_j}. \tag{10}$$

Note that (10) amounts to computing the ℓ_1 norm of a normalized version of vector $\boldsymbol{w}$. This induces *sparsity* in the final allocation. Also, the normalization with respect to F_j enhances the effectiveness of this ℓ_1 norm in providing solutions that are ℓ_0 optimal.

B. Resource Allocation Problem

A centralized Linear Program (LP) is formulated as follows,

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \gamma_1 f^E(\boldsymbol{x}) + \gamma_2 f^C(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{x} \in \textstyle\bigcap_{k=1}^6 \mathcal{X}_k \, , \, \boldsymbol{x} \geq \boldsymbol{0} \, , \end{aligned} \tag{11}$$

where $\gamma_1 \geq 0$ and $\gamma_2 \geq 0$ control the priority that is assigned to the minimization of the processing energy and to promoting local computations (if these are feasible), respectively.

V. Distributed allocation

Next, we use the ADMM method to derive a distributed algorithm that solves problem (11). In Section V-A, we show how to transform the centralized problem into a new form that lends itself to a distributed implementation, and in Section V-E we describe a distributed optimization scheme.

A. Problem transformation

As a first step towards a distributed implementation, we need the cost function $f^E(\boldsymbol{x})$ to be separable across servers, and therefore with respect to the partition $\boldsymbol{x} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_{\mathcal{M}_s})$. Ideally, we would like to get: $f^E(\boldsymbol{x}) = \sum_{i \in \mathcal{M}_s} f_i^E(\boldsymbol{x}_i)$. However, since server i does not know the incoming workload from its neighboring servers, it is unable to compute its own energy expenditure. To cope with this, for each server $i \in \mathcal{M}_s$ we introduce a *local* variable $\tilde{w}_{ji}^q$. Specifically, w_{ji}^q represents the actual outgoing workload of type q from server j to server i, whereas $\tilde{w}_{ji}^q$ is a *local estimate*, at server i, of the incoming workload from server j. The following consensus constraint is required to ensure consistency:

$$w_{ji}^q = \tilde{w}_{ji}^q \, . \tag{12}$$

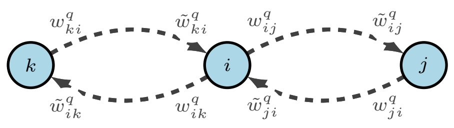


Fig. 2. Duplicated variables $\tilde{w}$ for server i and its neighbors j and k.

Because of (12), at optimality the duplicated variables are forced to take the same value. Therefore, we re-define the set of workload distribution variables $\boldsymbol{w}_i$, that are local to server i, by adding $\tilde{w}^q_{ji}$ for all $j \in \mathcal{N}_i$, $q \in \mathcal{Q}$ to vector $\boldsymbol{x}_i$. Note that w^q_{ii} does not require to be duplicated, since it is already local to server i. These constraints are expressed as:

$$\boldsymbol{x} \in \tilde{\mathcal{X}}_7 = \left\{\boldsymbol{x} : \; w^q_{ji} = \tilde{w}^q_{ji},\; i \;\in \mathcal{M}_s,\; j \in \mathcal{N}^*_i,\; q \in \mathcal{Q}\right\}. \tag{13}$$

Using these transformations, the objective function (9) can be written in a separable form as:

$$\begin{aligned}\tilde{f}^E_i(\boldsymbol{x}_i) &= c_i \left(\sum_{j\in\mathcal{N}^*_i} \sum_{q\in\mathcal{Q}} \frac{I_q \tilde{w}^q_{ji}}{2} + \sum_{q\in\mathcal{Q}} I_q w^q_{ii} \right) + \\ &\quad + \sum_{j\in\mathcal{N}^*_i} \sum_{q\in\mathcal{Q}} c_j \frac{I_q w^q_{ij}}{2} \,. \end{aligned} \tag{14}$$

In Fig. 2, we show an example of the just described duplication approach, for a network with three servers.

B. Local variables and constraints

Constraints can be categorized into two classes: local and consensus (or global). A constraint is said to be local to a server i if it only involves its own decision variables (contained in $\boldsymbol{x}_i$). Conversely, it is said to be a consensus constraint if it acts on variables that are spread across multiple servers. The constraint sets $\mathcal{X}^{(i)}_1$ and $\mathcal{X}^{(i)}_2$ are local to server $i \in \mathcal{M}_s$. $\mathcal{X}_3$, $\mathcal{X}_4$, $\mathcal{X}_6$, due to the new variables $\tilde{w}^q_{ji}$, can also be made local to server $i \in \mathcal{M}_s$, as follows:

$$\begin{aligned}\boldsymbol{x}_i \in \tilde{\mathcal{X}}^{(i)}_3 &= \{\boldsymbol{x}_i : I_q \tilde{w}^q_{ji} + \alpha^q_{ij} F_i (d_{ji} - \tau_q) \le 0\,, \\ &\qquad j \in \mathcal{M}_{s^{(1)}}, q \in \mathcal{Q}\}\,, \\ \boldsymbol{x}_i \in \tilde{\mathcal{X}}^{(i)}_4 &= \left\{ \boldsymbol{x}_i : \phi_i = \frac{1}{F_i} \sum_{j\in\mathcal{N}^*_i \cup \{i\}} \sum_{q\in\mathcal{Q}} \frac{I_q \tilde{w}^q_{ji}}{(\tau_q - d_{ji})} \right\}\,, \\ \boldsymbol{x}_i \in \tilde{\mathcal{X}}^{(i)}_6 &= \left\{\boldsymbol{x}_i : \; \alpha^q_{ij} \le \tilde{w}^q_{ji},\; j \in \mathcal{N}^*_i,\; q \in \mathcal{Q}\right\}\,. \end{aligned} \tag{15}$$

Instead, the constraint sets $\mathcal{X}_5$ and $\tilde{\mathcal{X}}_7$ are global constraints since they involve variables belonging to different servers. In Section V-D we show how to handle them.

C. Local objective functions

We use the local constraints $\mathcal{X}^{(i)}_1$, $\mathcal{X}^{(i)}_2$, and the local constraint sets in Eq. (15) by letting the local cost diverge to $+\infty$ for those points that are infeasible, i.e., outside the intersection of all local constraints. For server i, the new local objective function is thus defined as:

$$\begin{aligned}\tilde{f}_i(\boldsymbol{x}) &= \gamma_1 \tilde{f}^E_i(\boldsymbol{x}_i) + \gamma_2 f^C_i(\boldsymbol{x}_i)\,, \\ \mathrm{dom}\{\tilde{f}_i(\boldsymbol{x}_i)\} &= \{\mathcal{X}^{(i)}_1 \cap \mathcal{X}^{(i)}_2 \cap \tilde{\mathcal{X}}^{(i)}_3 \cap \tilde{\mathcal{X}}^{(i)}_4 \cap \tilde{\mathcal{X}}^{(i)}_6\}\,. \end{aligned} \tag{16}$$

D. Consensus variables and constraints

We now transform constraints involving variables belonging to different servers. The load-balancing constraints defined in Eq. (7), can be re-written through the addition of a new global variable z_ϕ:

$$-\frac{\theta}{2} \le \phi_i - z_\phi \le \frac{\theta}{2}, \quad \forall i \in \mathcal{M}_s\,. \tag{17}$$

The constraints $w^q_{ji} = \tilde{w}^q_{ji}$ can be transformed into a consensus form, through the introduction of the consensus variable z^q_{ji}. Hence, each server $i \in \mathcal{M}_s$ is subjected to the following consensus constraints:

$$\phi_i - z_\phi + u_i - \frac{\theta}{2} = 0\,, \tag{18a}$$

$$z_\phi - \phi_i + v_i - \frac{\theta}{2} = 0\,, \tag{18b}$$

$$w^q_{ij} - z^q_{ij} = 0,\; j \in \mathcal{N}^*_i,\; i \ne j,\; q \in \mathcal{Q}\,, \tag{18c}$$

$$\tilde{w}^q_{ji} - z^q_{ji} = 0,\; j \in \mathcal{N}^*_i,\; i \ne j,\; q \in \mathcal{Q}\,. \tag{18d}$$

Note that inequalities in (17) have been split, and two slack variables u_i and v_i have been added to $\boldsymbol{x}_i$ for every $i \in \mathcal{M}_s$ (see (18a) and (18b)). Constraints (18) are written in compact form as $\boldsymbol{P}_i\boldsymbol{x}_i + \boldsymbol{Q}_i\boldsymbol{z} + \boldsymbol{m} = \boldsymbol{0}$, for every server $i \in \mathcal{M}_s$, where matrices $\boldsymbol{P}_i$ and $\boldsymbol{Q}_i$ have elements in $\{0, -1, 1\}$, whereas vector $\boldsymbol{m}$ has elements in $\{0, -\theta/2\}$.

E. ADMM-based algorithm

We are now ready to write the problem in a distributed form:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \sum_{i\in\mathcal{M}_s} f_i(\boldsymbol{x}_i) \\ \text{s.t.} \quad & \boldsymbol{P}_i\boldsymbol{x}_i + \boldsymbol{Q}_i\boldsymbol{z} + \boldsymbol{m} = \boldsymbol{0}, \quad i \in \mathcal{M}_s, \\ & \boldsymbol{x}_i \ge \boldsymbol{0}, \quad i \in \mathcal{M}_s\,. \end{aligned} \tag{19}$$

This problem has a separable objective function across the servers, and the global variables $\boldsymbol{z}$ (being shared by different servers) make the problem non fully separable. To derive a fully distributed algorithm, we define the augmented Lagrangian associated with problem (19):

$$\begin{aligned} L_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda}) = \sum_{i\in\mathcal{M}_s} \Bigg(& f_i(\boldsymbol{x}_i) + \boldsymbol{\lambda}^T_i \left(\boldsymbol{P}_i\boldsymbol{x}_i + \boldsymbol{Q}_i\boldsymbol{z} + \boldsymbol{m}\right) + \\ & + \frac{\rho}{2} ||\boldsymbol{P}_i\boldsymbol{x}_i + \boldsymbol{Q}_i\boldsymbol{z} + \boldsymbol{m}||^2 \Bigg), \end{aligned} \tag{20}$$

where ρ is the augmentation parameter, $\boldsymbol{\lambda} = (\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_{\mathcal{M}_s})$ represents the dual variable vector, whose i-th component $\boldsymbol{\lambda}_i = (\lambda^u_i, \lambda^v_i, \{\lambda^q_{ij}, \tilde{\lambda}^q_{ji}\}_{j\in\mathcal{N}^*_i, q\in\mathcal{Q}})$ collects the dual variables: λ^u_i, λ^v_i, λ^q_{ij} and $\tilde{\lambda}^q_{ji}$ associated with the constraints in (18). Note that $L_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda}) = \sum_{i\in\mathcal{M}_s} L^{(i)}_\rho(\boldsymbol{x}_i, \boldsymbol{z}, \boldsymbol{\lambda}_i)$. The three-step ADMM algorithm shown in Algorithm 1 is derived through the minimization of the augmented Lagrangian with respect to the $\boldsymbol{x}_i$ variables, which decompose into $\mathcal{M}_s$ separate quadratic convex problems, the minimization of the consensus (global) variables $\boldsymbol{z}$, and the dual-variable $\boldsymbol{\lambda}$ update.

Algorithm 1 ADMM

```
1: $\boldsymbol{x}_i^{k+1} = \text{argmin}_{\boldsymbol{x}_i, \boldsymbol{x}_i \geq \boldsymbol{0}} L_\rho^{(i)}(\boldsymbol{x}_i, \boldsymbol{z}^k, \boldsymbol{\lambda}_i^k)$
2: $\boldsymbol{z}^{k+1} = \text{argmin}_{\boldsymbol{z}} L_\rho(\boldsymbol{x}^{k+1}, \boldsymbol{z}, \boldsymbol{\lambda}^k)$
3: $\boldsymbol{\lambda}_i^{k+1} = \boldsymbol{\lambda}_i^k + \rho\left(\boldsymbol{P}_i \boldsymbol{x}_i^{k+1} + \boldsymbol{Q}_i \boldsymbol{z}^{k+1} + \boldsymbol{m}\right)$
```

Note that each server i can independently update $\boldsymbol{x}_i$ (step 1 of Algorithm 1) and also the dual variables $\boldsymbol{\lambda}_i$ (line 3). The z-minimization step (step 2) can be further simplified and written in closed form. The problem to be solved is:

$$z_\phi^{k+1} = \underset{z_\phi}{\text{argmin}} \sum_{i \in \mathcal{M}_s} \Bigg[\lambda_i^{u,k} \left(\phi_i^{k+1} - z_\phi + u_i^{k+1} - \frac{\theta}{2} \right) + \\ + \lambda_i^{v,k} \left(z_\phi - \phi_i^{k+1} + v_i^{k+1} - \frac{\theta}{2} \right) + \\ + \frac{\rho}{2} \left(\left(\phi_i^{k+1} - z_\phi + u_i^{k+1} - \frac{\theta}{2} \right)^2 + \left(z_\phi - \phi_i^{k+1} + v_i^{k+1} - \frac{\theta}{2} \right)^2 \right) \Bigg] . \tag{21}$$

Once solved, (21) leads to:

$$z_\phi^{k+1} = \frac{1}{|\mathcal{M}_s|} \sum_{i \in \mathcal{M}_s} \left(\phi_i^{k+1} + \frac{u_i^{k+1} - v_i^{k+1}}{2} + \frac{\lambda_i^{u,k} - \lambda_i^{v,k}}{2\rho} \right) . \tag{22}$$

(22) can be simplified by noting that after the first iteration we have $\sum_{i \in \mathcal{M}_s} \left(\lambda_i^{v,k+1} - \lambda_i^{u,k+1} \right) = 0$, leading to:

$$z_\phi^{k+1} = \frac{1}{|\mathcal{M}_s|} \sum_{i \in \mathcal{M}_s} \left(\phi_i^{k+1} + \frac{u_i^{k+1} - v_i^{k+1}}{2} \right) . \tag{23}$$

Now, we derive the closed-form update for $z_{ij}^{q,k+1}$, for each $i \in \mathcal{M}_s,\ j \in \mathcal{N}_i,\ i \neq j$:

$$z_{ij}^{q,k+1} = \underset{z_{ij}^q}{\text{argmin}}\, \lambda_{ij}^{q,k} \left(w_{ij}^{q,k+1} - z_{ij}^q \right) + \tilde{\lambda}_{ij}^{q,k} \left(\tilde{w}_{ij}^{q,k+1} - z_{ij}^q \right) + \\ + \frac{\rho}{2} \left(\left(w_{ij}^{q,k+1} - z_{ij}^q \right)^2 + \left(\tilde{w}_{ij}^{q,k+1} - z_{ij}^q \right)^2 \right) , \tag{24}$$

whose solution is: $z_{ij}^{q,k+1} = (w_{ij}^{q,k+1} + \tilde{w}_{ij}^{q,k+1})/2 + (\lambda_{ij}^{q,k} + \tilde{\lambda}_{ij}^{q,k})/(2\rho)$. Also in this case, we observe that the Lagrangian multiplier term becomes zero after the first iteration (since $\lambda_{ij}^{q,k+1} + \tilde{\lambda}_{ij}^{q,k+1} = 0$), thus, we can simplify the update equations $z_{ij}^{q,k+1}$ as:

$$z_{ij}^{q,k+1} = \frac{w_{ij}^{q,k+1} + \tilde{w}_{ij}^{q,k+1}}{2} . \tag{25}$$

Note that the update $\boldsymbol{z}^{k+1}$ can be compactly written as a linear transformation of vector $\boldsymbol{x}$, i.e., $\boldsymbol{z}^{k+1} = \boldsymbol{S}\boldsymbol{x}^{k+1}$, where matrix $\boldsymbol{S}$ is constructed according to (23) and (25).

The final ADMM algorithm is reported in Algorithm 2.

Algorithm 2 Final ADMM algorithm

```
1: $\boldsymbol{x}_i^{k+1} = \text{argmin}_{\boldsymbol{x}_i} L_\rho^{(i)}(\boldsymbol{x}_i, \boldsymbol{z}^k, \boldsymbol{\lambda}_i^k)$
2: $\boldsymbol{z}^{k+1} = \boldsymbol{S}\boldsymbol{x}^{k+1}$
3: $\boldsymbol{\lambda}_i^{k+1} = \boldsymbol{\lambda}_i^k + \rho\left(\boldsymbol{P}_i \boldsymbol{x}_i^{k+1} + \boldsymbol{Q}_i \boldsymbol{z}^{k+1} + \boldsymbol{m}\right)$
```

F. Communications among agents and stopping criterion

Algorithm 2 entails the following: a local computation effort is needed for the update in step 1, step 2 instead requires each server i to broadcast $\boldsymbol{x}_i^{k+1}$ to all network nodes. This is required by the load balancing constraint (17), which demands all loads to distribute around a common mean value. Step 3 is another local processing effort. To evaluate the algorithm's convergence, we can compute the residual at each iteration. As suggested in [2], we can exploit the primal and dual residuals, which are respectively: $\boldsymbol{r}^{k+1} = \boldsymbol{P}\boldsymbol{x}^{k+1} + \boldsymbol{Q}\boldsymbol{z}^{k+1} + \boldsymbol{m}$ and $\boldsymbol{s}^{k+1} = \rho \boldsymbol{P}^T \boldsymbol{Q} \left(\boldsymbol{z}^{k+1} - \boldsymbol{z}^k \right)$. The matrices $\boldsymbol{Q}$ and $\boldsymbol{P}$ trivially descend from $\boldsymbol{Q}_i$ and $\boldsymbol{P}_i$ to match the full vector $\boldsymbol{x}$. As proposed in [2], the iterative algorithm shall stop when the two residuals fall below two dynamically computed thresholds, i.e., when both inequalities $||\boldsymbol{r}^k|| \leq \epsilon_{\text{pri}}^k$ and $||\boldsymbol{s}^k|| \leq \epsilon_{\text{dual}}^k$ are verified. These dynamic thresholds are defined through absolute and relative tolerances: ϵ^{abs} and ϵ^{rel} as follows:

$$\begin{aligned} \epsilon_{\text{pri}}^k &= \sqrt{(\text{len}(\boldsymbol{\lambda})}\epsilon^{\text{abs}} + \epsilon^{\text{rel}} \max\{||\boldsymbol{A}\boldsymbol{x}^k||, ||\boldsymbol{Q}\boldsymbol{z}^k||, ||\boldsymbol{m}||\} \\ \epsilon_{\text{dual}}^k &= \sqrt{(\text{len}(\boldsymbol{x})}\epsilon^{\text{abs}} + \epsilon^{\text{rel}} ||\boldsymbol{P}^T \boldsymbol{\lambda}^k|| . \end{aligned} \tag{26}$$

VI. Numerical Results

We structure the numerical results section into two parts: a toy-example network to illustrate the flexibility offered by our centralized framework presented in Section IV, and a more complex scenario to test the convergence properties of the decentralized algorithm proposed in Section V.

Toy-network: We consider a scenario with $M_s = 5$ servers, connected according to a mesh topology and organized in two tiers. Servers $\{1, 2, 3, 4\}$ belong to the first tier and therefore collect the workload generated at the associated BSs. Server 5 is in the second tier. The processing requests collected by the first tier MEC servers are of two types $\mathcal{Q} = \{1, 2\}$. The first application type ($q = 1$) denotes a highly time-sensitive application, e.g., near real-time video recognition. The second application ($q = 2$) is more delay tolerant than $q = 1$. For example, it could be some kind of non real-time voice or video processing task.

The processing capability of each server F_i, expressed in Mega CPU (MCPU) cycles per millisecond are: $(F_1, F_2, F_3, F_4, F_5) = (6, 9, 10, 8, 40)$. Moreover, the execution cost per MCPU cycle c_i incurred by the servers is: $(c_1, c_2, c_3, c_4, c_5) = (1.2, 0.9, 0.9, 1.3, 2)$. The latency introduced by the communication among servers belonging to the first tier is 5 ms, whereas transmitting data between servers belonging to different tiers incurs a latency of 15 ms. The task completion deadlines associated with the two application types are $(\tau_1, \tau_2) = (20, 100)$ milliseconds. The workload that each server has to handle is collected in the vector $(W_1^1, W_1^2, W_2^1, W_2^2, \ldots, W_5^1, W_5^2) = (40, 300, 110, 300, 20, 100, 120, 100, 0, 0)$ Mbit. Moreover, the number of CPU cycles per bit is fixed to 1 for both applications, i.e., $(I_1, I_2) = (1, 1)$. This can be thought as a non-congested network, as all the servers have enough processing power to handle all the gathered workload *locally*, without allocating it to nearby servers.

In Fig. 3, we show the results obtained for some selected parameters γ_1, γ_2 and θ. The final workload allocations ϕ_i

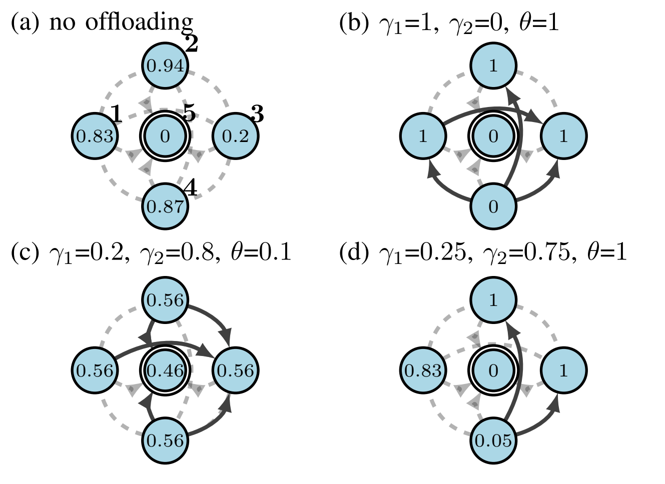
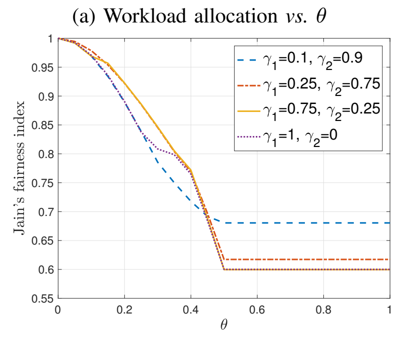

(a) no offloading
(b) γ_1=1, γ_2=0, θ=1
(c) γ_1=0.2, γ_2=0.8, θ=0.1
(d) γ_1=0.25, γ_2=0.75, θ=1

Fig. 3. Values inside the circles represent the load factors ϕ_i, and bold numbers in (a) represent servers' ids. Network's workload allocation when: (a) no offloading is allowed (local processing only), (b) offloading is allowed, but neither cooperation control ($\gamma_2 = 0$) nor load balancing ($\theta = 1$) are performed, (c) cooperation control, load balancing and energy consumption minimization are used, (d) only energy consumption minimization and cooperation control are applied.



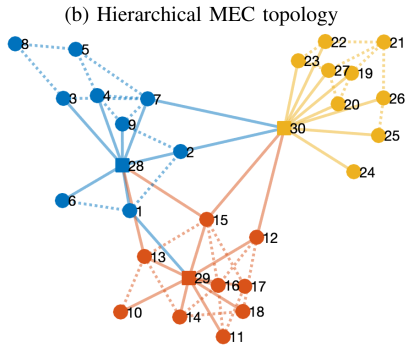


Fig. 4. (a) Jain's fair index evaluated on the workload distribution *vs* θ. (b) Hierarchical MEC architecture. 1^{st}-layer ($\mathcal{M}_{s(1)}$) and 2^{st}-layer ($\mathcal{M}_{s(2)}$) nodes are respectively represented by squares and circles. The color partitions the set $\mathcal{M}_{s(1)}$ into $|\mathcal{M}_{s(2)}|$ clusters. Dotted and solid edges respectively represent intra-cluster and cross-layer communication links.

(see Eq. (6)) are indicated inside the circles, which denote the servers. Fig. 3(a) shows the workload allocation when no offloading (cooperation) is performed. In this case, the edge servers in the first tier process their tasks *locally* and the server in the second tier (in the center) is not used. Fig. 3(b) refers to the case where offloading is allowed, but neither cooperation control (via the cost term $f_i^C(\boldsymbol{x}_i)$, associated with γ_2) nor load balancing (via Eq. (17)) are enforced. In this case, the objective solely corresponds to minimizing the energy consumed by the whole processing network, and the workload is allotted to the servers having the lowest cost per CPU cycle. Fig. 3(c) refers to the case where cooperation control, load balancing and power consumption minimization are jointly applied. In this case, a fairer distribution of the workload is obtained. To reach this, some of the workload is sent to both high cost and second tier servers. The last Fig. 3(d) shows the workload allocation when cooperation control (sparsity) and power consumption minimization are enforced. From Fig. 3(d) we observe that the cooperation control weight (γ_2) leads to a low number of re-allocated tasks (two in Fig. 3(d), represented by the solid black arrows). This does not happen for Fig. 3(c) as the load balancing parameter for that plot is $\theta = 0.1$, which acts as a strict constraint on the load factors, allowing a maximum deviation of $\theta = 0.1$ among the load factors ϕ_i of any two servers.

To assess the load balancing capability of our optimization framework, we use the Jain's Fairness Index, defined as:

$$J(\boldsymbol{\phi}) = \frac{\left(\sum_{i\in\mathcal{M}_s}\phi_i\right)^2}{|\mathcal{M}_s|\sum_{i\in\mathcal{M}_s}\phi_i^2}\,, \tag{27}$$

where $\boldsymbol{\phi} = \{\phi_i\}_{i\in\mathcal{M}_s}$.

In Fig. 4a, we plot $J(\boldsymbol{\phi})$ as a function of the load balancing parameter θ, for some selected values of γ_1 and $\gamma_2 = 1-\gamma_1$. As expected, constraint (17) is an effective approach for balancing the workload allocation across servers: as $\theta \to 0$ the servers tend to allocate the same amount of workload, i.e., the load factors ϕ_i tend to become all equal to one another. Conversely, as $\theta \to 1$, constraint (17) becomes inactive and the workload distribution is decided based on other costs, namely, energy $f_i^E(\boldsymbol{x}_i)$ and cooperation control $f_i^C(\boldsymbol{x}_i)$.

Distributed network: In the second scenario, we consider a network topology involving a larger number of nodes ($|\mathcal{M}_s| = 30$). In Fig. 4b we report an instance of such network. Second-layer and first-layer nodes are respectively represented by squares and circles. The higher nodes induce a partition on the underlying ones, which is encoded through colors. A connection between two first tier nodes that fall within the same cluster exists with probability $p_{11} = 0.3$. Moreover, any first layer node is connected with its second-layer node (square, same color) and with any other layer-2 node with probabilities $p_{12} = 0.9$ and $p_{12}^{\text{cross}} = 0.1$, respectively. Let us indicate with $i, j \in \mathcal{M}_{s(1)}$ two nodes belonging to the same cluster, with k the associated second-layer node and with $l \in \mathcal{M}_{s(2)}$ a generic second-layer node. We define the delays (in millisecond) as $d_{ij} \sim \mathcal{U}([5, 25])$, $d_{ik} \sim \mathcal{U}([30, 70])$, and $d_{il} \sim \mathcal{U}([50, 110])$ for $l \neq k$. The computational capacity of each node F_i expressed in MCPU cycles per millisecond is generated according to $\mathcal{U}([7, 9])$ and $\mathcal{U}([12, 15])$, for $i \in \mathcal{M}_{s(1)}$ and $i \in \mathcal{M}_{s(2)}$ respectively. In the same way, the energy cost per MCPU cycle is drawn from the following two distributions $\mathcal{U}([0.6, 1])$ and $\mathcal{U}([1.2, 1.9])$. The workload of types $q = 1$ and $q = 2$ collected by each first-layer node and measured in Mbit is generated according to the following two distributions $\mathcal{U}([100, 120])$ and $\mathcal{U}([200, 300])$. In Fig. 5, we show some example results for the distributed ADMM algorithm. The used parameters are $\gamma_1 = 0.8$ and $\gamma_2 = 0.2$ In Fig. 5a and 5b the convergence behavior of the objective function with respect to the optimal (centralized) one, the primal and dual residuals, and the primal and dual thresholds

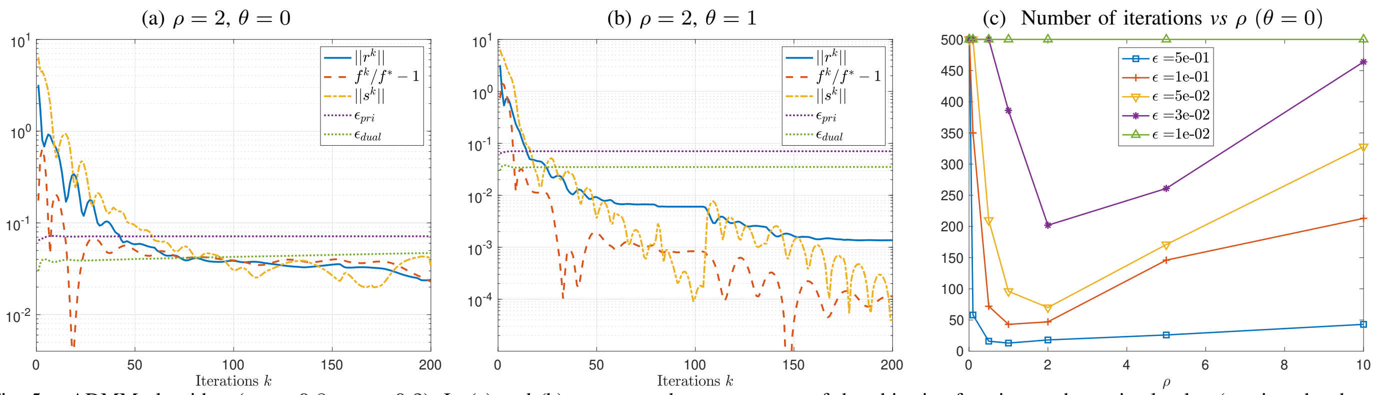


Fig. 5. ADMM algorithm ($\gamma_1 = 0.8$, $\gamma_2 = 0.2$). In (a) and (b) we report the convergence of the objective function to the optimal value (as given by the centralized solution f^*), the convergence of the primal and dual residual norms ($||\boldsymbol{r}^k||$, $||\boldsymbol{s}^k||$), for $\epsilon = 0.001$, and the thresholds ($\epsilon_{\rm pri}$, $\epsilon_{\rm dual}$) for $\theta = \{0, 1\}$. In (c) the number of iterations required to converge with a given accuracy ($\epsilon^{\rm abs} = \epsilon^{\rm rel} = \epsilon$) is shown, as a function of the penalty factor ρ, with $\theta = 0$.

are plotted as a function of the number of iterations k, for two choices of the load balancing parameter $\theta = \{0, 1\}$. For $\rho = 2$ and $\epsilon = 0.001$, convergence occurs within 100 and 50 iterations when $\theta = 0$ and $\theta = 1$, respectively. Finally, in Fig. 5c we plot the number of iterations to converge for several values of the augmentation parameter ρ, which greatly affects the convergence speed of the distributed algorithm. As expected, the number of iterations that the decentralized algorithm requires to converge increases as ϵ becomes smaller. The case $\theta = 1$ entails a similar performance tradeoff, but leads to a higher precision within fewer iterations (the plot is omitted in the interest of space).

VII. Conclusions

In this work, we have proposed an optimization approach for the allocation of processing tasks in hierarchical MEC networks. Our model deals with contrasting objectives, and optimal algorithms are formulated in centralized and distributed settings. ADMM offers advantages such as the problem decomposition into smaller local subproblems, which are easier to solve than the original (centralized) one. Our framework can be extended in several ways: 1) the ADMM algorithm can be accelerated, 2) besides parallel processing, the sequential processing of workload at the servers can also be modeled, 3) the optimization can be adapted to a dynamic setting, where computing requests are stochastic and arrive at the BSs according to some statistics. This amounts to dealing with backlogs and new job arrivals and the integration of these facts with a distributed algorithm is non trivial.

Acknowledgment

This work has received funding from the Italian Ministry of Education, University and Research (MIUR) through the PRIN project no. 2017NS9FEY entitled “Realtime Control of 5G Wireless Networks: Taming the Complexity of Future Transmission and Computation Challenges”.

References

[1] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, “Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges,” *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54–61, 2017.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers,” *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011.

[3] 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.

[4] E. Baccarelli, N. Cordeschi, A. Mei, M. Panella, M. Shojafar, and J. Stefa, “Energy-efficient dynamic traffic offloading and reconfiguration of networked data centers for big data stream mobile computing: review, challenges, and a case study,” *IEEE Network*, vol. 30, no. 2, pp. 54–61, March 2016.

[5] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, “Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems,” *IEEE Trans. on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.

[6] J. Oueis, E. C. Strinati, and S. Barbarossa, “The fog balancing: Load distribution for small cell cloud computing,” in *IEEE Vehicular Technology Conference (VTC Spring)*, Glasgow, UK, May 2015.

[7] ——, “Distributed mobile cloud computing: A multi-user clustering solution,” in *IEEE International Conference on Communications (ICC)*, Kuala Lumpur, Malaysia, May 2016.

[8] L. Chen, J. Xu, and S. Zhou, “Computation Peer Offloading in Mobile Edge Computing with Energy Budgets,” in *IEEE GLOBECOM*, Singapore, Dec. 2017.

[9] W. Fan, Y. Liu, B. Tang, F. Wu, and Z. Wang, “Computation Offloading Based on Cooperations of Mobile Edge Computing-Enabled Base Stations,” *IEEE Access*, vol. 6, pp. 22 622–22 633, 2017.

[10] Q. Fan and N. Ansari, “Workload Allocation in Hierarchical Cloudlet Networks,” *IEEE Communications Letters*, vol. 22, no. 4, pp. 820–823, 2018.

[11] J. Xu, L. Chen, and S. Ren, “Online learning for offloading and autoscaling in energy harvesting mobile edge computing,” *IEEE Trans. on Cognitive Communications and Networking*, vol. 3, no. 3, pp. 361–373, 2017.

[12] A. Kiani and N. Ansari, “Toward hierarchical mobile edge computing: An auction-based profit maximization approach,” *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 2082–2091, 2017.

[13] H. Guo, J. Liu, H. Qin, and H. Zhang, “Collaborative Computation Offloading for Mobile-Edge Computing over Fiber-Wireless Networks,” in *IEEE GLOBECOM*, Singapore, Dec. 2017.

[14] Y. Zhou, F. R. Yu, J. Chen, and Y. Kuo, “Resource Allocation for Information-Centric Virtualized Heterogeneous Networks With In-Network Caching and Mobile Edge Computing,” *IEEE Trans. on Vehicular Technology*, vol. 66, no. 12, pp. 11 339–11 351, 2017.

[15] T. Yang, H. Zhang, H. Ji, and X. Li, “Computation collaboration in ultra dense network integrated with mobile edge computing,” in *IEEE International Annual Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Montreal, QC, Canada, Oct. 2017.

[16] X. Lyu, H. Tian, C. Sengul, and P. Zhang, “Multiuser joint task offloading and resource optimization in proximate clouds,” *IEEE Trans. on Vehicular Technology*, vol. 66, no. 4, pp. 3435–3447, 2017.

[17] X. Chen, L. Jiao, W. Li, and X. Fu, “Efficient multi-user computation offloading for mobile-edge cloud computing,” *IEEE/ACM Trans. on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.