Jointly Learning Optimal Task Offloading and Scheduling Policies for Mobile Edge Computing

Livia Elena Chatzieleftheriou[1] and Iordanis Koutsopoulos[2]
[1] IMDEA Networks Institute, Madrid, Spain
[2] Department of Informatics, Athens University of Economics and Business, Athens, Greece

***Abstract*—This work contributes towards optimizing edge analytics in Mobile Edge Computing (MEC) systems. We consider requests for computing tasks that are generated from users and can be satisfied either locally at their devices, or they can be offloaded to an edge server in their proximity for remote execution. We study a multi-user MEC system with limited energy autonomy for the mobile devices and with limitations on the computing capability of both mobile devices and at an edge server, where users can offload part of their computation load. We define a utility over "resource residuals", that capture the difference between the resources assigned through our decisions, and those needed in practice, and we aim at the minimization of regret, *i.e.*, of the difference between the utility obtained by an optimal offline benchmark that knows the system evolution in hindsight, and our online decision policy. We design an algorithm that jointly learns policies for offloading computations and scheduling them for execution at the shared MEC server. We prove that our algorithm is asymptotically optimal, *i.e.*, it has no regret over the optimal static offline benchmark, and that its performance is independent of the number of devices in the system. From our numerical evaluation we conclude that our algorithm adapts to unpredictable demand changes, it learns to identify resource-limited devices, and it learns to share the server's resources.**

I. Introduction

Emerging technologies, such as Augmented Reality (AR) or Internet of Things (IoT), rely on delay-intolerant and computationally-intense operations, *e.g.*, image recognition or data analytics. The computing and energy autonomy of mobile devices is still limited and these opera-tions are too complex to be executed entirely on mobile devices [1]. In the Mobile Edge Computing (MEC) paradigm, computing servers are introduced at the network edge, providing to the users the opportunity to offload part of their computation load for remote execution. These servers have limited computation capacity that must be shared among the computing tasks of different users [2].

To have good resource utilization, a careful allocation of the limited computation and energy resources at the network edge is needed. Thus, aiming at a better utilization of the existing limited resources for a given computation demand, two questions arise: (i) What portion of each user's computation demand should be offloaded for remote execution to the server? (ii) How should the offloaded tasks be scheduled at the server's shared computing facility? There is evidence that the computational demand is highly unpredictable [3], This could happen, *e.g.*, due to changes in the users' needs, or due to changes in the user-to-server association. In practice, this unpredictability complicates the accurate statistical characterization of the arising computing load, and precludes the use of approaches that operate under stationary regimes, such as Lyapunov optimization. The simultaneous need of jointly deciding both the portion of tasks that users will offload and their scheduling at the server, without relying on accurate statistics for the computation demand, makes the good utilisation of resources very challenging.

We overcome these challenges by *learning to jointly offload and schedule computation tasks, without any assumption on the rate with which computation task requests arise at each node.* We consider energy and computation limitations for mobile devices and computing limitations at the edge server. Offloading reflects the portion of tasks that users offload for remote execution, and *assign* computation load to the server. Scheduling reflects the portion of computing resources of the edge server that are *reserved* for each user. Offloading and scheduling decisions should assign computing load that is at most as high as the reserved computing resources. We consider a similar quantity for the users' energy resources. The residuals' role is crucial: large residuals allow the system to handle computation demand that may appear unpredictably. We define a system utility over the above residuals. To maximize the utility for known computation demand, we study the offline Joint Computation Offloading and Scheduling Problem (JCOSP). We then define its online instance (for unknown demand), aiming at the regret minimization *i.e.*, the difference between the utility obtained by online decisions and that obtained by an optimal static policy that knows computation demand patterns in hindsight. We design a simple and fast online algorithm that jointly learns which tasks to offload and how to schedule them at the shared edge server, without making any assumption on the computation demand pattern of arrival rate. We prove that it is asymptotically optimal w.r.t. the benchmark, scalable, and able to adapt to unpredictable changes in computation demand.

In Section II we discuss related work. In Sections III and IV we present our setting and our optimization problems. In Sections V and VI we design and evaluate our algorithm, both analytically and numerically. In Section VII we conclude the paper.

Our contributions to the literature are as follows:

- We formulate the Joint Computation Offloading and task Scheduling Problem (JCOSP), under computation, energy and bandwidth limitations at the network edge. Our objective utility function promotes prudent resource allocation through maximizing a utility derived from residuals, to accommodate computation demand that may appear unpredictably.
- We identify and exploit inherent properties of the JCOSP,

The work of Livia Elena Chatzieleftheriou was done while she was affiliated with the Athens University of Economics and Business, Athens, Greece.

e.g., concavity w.r.t. decisions. We design OJOSO, a simple algorithm that learns to adapt the scheduling and offloading policies to unpredictable demand changes, without making assumptions on the pattern of arrival rate of the computation demand. We prove its no-regret property and its scalability.
- We evaluate OJOSO over datasets explicitly generated to obstruct its decisions, highlighting its optimal behaviour: OJOSO learns to identify computationally- and energy-limited devices, to optimally share the server's resources among users, and learns the optimal amount of computations to execute locally at the devices and remotely at the edge server.

II. Related work

Jointly deciding scheduling and computation offloading. In [4], a linear optimization problem for energy consumption minimization under delay constraints, deadlines and task-dependency is formulated and solved optimally with standard optimization techniques. The work [5] aims at minimizing a sum of energy and latency under transmission power constraints for one user. For their non-convex mixed-integer optimization problem, the authors propose a two-level alternation method based on Lagrangian dual decomposition. In [6] energy consumption minimization under computation latency constraints is studied, additionally deciding the devices' transmission power and processing speed. The problem is optimally solved by convexification. Despite giving interesting results, these works look at static instances of the joint problem. Our work accounts for the *time evolution of the system* and considers an *adversarial approach to mitigate the lack of information* about the amount of computation demand when deciding, which is common in scenarios with a massive number of devices requesting computation, like AR.

Online Convex Optimization (OCO) in MEC. In [7] a periodic benchmark for online learning is introduced and a no-regret algorithm for online user association is presented. In [8] computation offloading for IoT under communication, power and computation constraints is studied. We aim at *maximizing a utility derived from residual computation and energy* resources in MEC.

III. Model

We describe our model components and define the "residual resources", which are fundamental components of our analysis.

A. Model components and decision variables

We consider a set $\mathcal{U}$ of user devices and one MEC server M, that operate at f_u and f_M CPU-cycles/sec, respectively.

Time-dynamics. We consider a time horizon T that is partitioned into T discrete slots, *i.e.*, $t = 1, \dots, T$. Within each slot t all values and decisions are assumed fixed, while they may change from one slot to the other. In practice, the duration s of slots may be in the order of secs, so that it captures the sensitivity of most apps, *e.g.*, AR, and it is sufficient for computations to be executed.

Tasks. The users' activity generates computing tasks over some input data, *e.g.*, the Microsoft "Seeing AI" app performs object recognition over users' photos. Tasks are finite sets of computations over the input, all of which are needed for obtaining a meaningful result, *e.g.*, to correctly identify the content of the photo. Tasks are characterized by their computational load l (in CPU-cycles/task) and the size d of the input (in bits/task). The number $\lambda_u(t)$ of tasks generated by user u during slot t is arbitrary [3]. Let vector $\boldsymbol{\lambda}(t) = \{\lambda_u(t)\}_{u\in\mathcal{U}}$. Tasks must be executed within the slot they were generated and, without loss of generality (w.l.o.g.), tasks are generated at the beginning of the corresponding slot.

Computation offloading. Tasks can be executed either entirely locally at the users' devices, or they can be entirely offloaded to the server for remote execution (transmitting the input data there). In the envisioned B5G/6G networks, the transmission rates for the user-to-server communication are on the order of Gbps [2]. Existing commercial cloud-based AR applications offload images of average size of $d = 73.56$KB [1], resulting in transmission delays $< 10^{-3}$ secs. Moreover, B5G/6G networks are dense, *i.e.*, propagation delays are negligible compared to transmission delays. Also, it takes on average longer than 2 seconds to finish object recognition on a mobile CPU [9]. We thus consider only execution delays, since transmission and propagation delays are negligible compared to them. We assume offloading decisions taken by a central agent, who dictates the offloading policy to devices. Let $x_u(t) \in [0, 1]$ be the portion of tasks that user u should offload during slot t; thus u offloads $\lambda_u(t)x_u(t)$ tasks during t. The remaining portion $(1 - x_u(t))$ of tasks are executed locally. Let $\mathbf{x} = \{x_u(t)\}_{u\in\mathcal{U}}$ be the offloading policy.

Energy consumption. We consider the energy that the users' devices spend for: (a) locally executing tasks, and (b) transmitting data to the server. The energy consumption for locally executing k CPU-cycles equals $c_{u1}kf_u^2$ Watt-hours (Wh) [10], where $c_{u1} > 0$ a hardware-related constant. Data transmission consumes energy proportional to the amount of data transmitted. The total energy $\epsilon_u\big(\lambda_u(t), x_u(t)\big)$ consumed by u during t (in Wh) depends on the number $\lambda_u(t)$ of generated tasks and her offloading policy $x_u(t)$. It is the sum of the energy spent for transmission and for computation. For c_{u2} a transmission-related constant, the total energy consumption $\epsilon_u\big(\lambda_{u(t)}, x_u(t)\big)$ for user u equals:

$$\epsilon_u\big(\lambda_u(t), x_u(t)\big) = c_{u1}f_u^2 l\lambda_u(t)\big(1 - x_u(t)\big) + c_{u2}d\lambda_u(t)x_u(t). \tag{1}$$

The users' batteries devices may spend a total amount of energy during the entire time horizon T, leading to "budget" constraints that bound the energy consumption $\sum_{t=1}^{T}\epsilon_u\big(\lambda_u(t), x_u(t)\big)$ over T. However, online scenarios under budget constraints cannot be solved asymptotically optimally [11]. We consider a strict energy consumption constraints, allowing our system to use a maximum amount of e_u Wh/slot in each device. This captures a strict bound on the average energy *spent per slot* assuming a steady state. Observe that strict energy constraints are more conservative than average ones, *i.e.*, when the strict constraint is satisfied, we ensure that the average budget constraint is satisfied as well.

Scheduling. It captures the portion of the computation capacity to allocate at each of the offloaded tasks. At each slot t, the server decides the scheduling policy $\mathbf{y}(t) = \{y_u(t)\}_{u\in\mathcal{U}}$, where $y_u(t)$ reflects the portion of the server's total computing capacity f_M that is allocated to the tasks that user u offloaded. We assume that the server applies the Generalized Processor Sharing (GPS) scheduler [12], due to its simplicity and guaranteed rates. Given

y_u, the GPS scheduler guarantees to each user u the execution of $y_u f_M$ CPU-cycles/sec. Without loss of generality, let

$$\sum_{u\in\mathcal{U}} y_u(t) = 1,\ y_u \in [0,1], \forall t = 1,\dots,T.$$

B. Residual resources and system utility

Residual resources. Scheduling *reserves* computing resources at the server for executing each user's offloaded tasks. Offloading *assigns* a portion of each user's computation load for remote execution, and *determines* her energy consumption. These decisions are taken and applied at the same slot. With a high margin between the needed and the allocated resources, we can better accommodate computation demand that may arise unpredictably.

Let $D_{Mu}(t)$ (in CPU-cycles/slot) be the difference between the computing resources that were reserved by scheduling decisions at the server for user u, and those assigned by offloading decisions from u to the server. Let $D_{uc}(t)$ (in CPU-cycles/slot) be the difference between the computation resources sf_u that are available at device u, and the assigned computation load due to offloading decision for u. Let $D_{ue}(t)$ (in Wh/slot) be the difference between the energy resources e_u that are available at u, and the assigned energy consumption due to the offloading decision for u.

All residuals, *i.e.*, energy and computation, are based on offloading and scheduling decisions taken during *the same time slot*. They are essential for online scenarios, where the computation demand is not known in advance. Indeed, the allocated resources may substantially vary from those needed, since they are decided under lack of information about the actual needs. Positive residuals imply resource under-utilization, while negative residuals mean resource over-utilization, and must be avoided. We formally have:

$$D_{Mu}\big(\lambda_u(t), x_u(t), y_u(t)\big) := sf_M y_u(t) - l\lambda_u(t)x_u(t), \quad (2a)$$

$$D_{uc}\big(\lambda_u(t), x_u(t)\big) := sf_u - l\lambda_u(t)\big(1 - x_u(t)\big), \quad (2b)$$

$$D_{ue}\big(\lambda_u(t), x_u(t)\big) := e_u - \epsilon_u\big(\lambda_u(t), x_u(t)\big), \quad (2c)$$

where s is the slot duration and f_M is the clock frequency (in CPU-cycles per second) at M.

Utility function desiderata. We aim at better accommodating computation load that may appear unexpectedly. This implies leaving as many computation resource residuals $D_{Mu}(\cdot), D_{uc}(\cdot)$, and energy resource residuals $D_{ue}(\cdot)$ as possible, through an appropriate combination of offloading and server scheduling policies. Utility functions $g(\cdot)$ need to satisfy the following properties:

Assumption 1. *The utility $g(D)$ takes as input the residuals $D \in \{D_{Mu}(\cdot), D_{uc}(\cdot), D_{ue}(\cdot)\}$. The utility $g(D)$ is monotone increasing, concave, and Lipschitz-continuous w.r.t. the residual amount of resource, D.*

Monotonicity promotes using resources only when necessary. Additional resource units are more important when residuals are lower, to avoid them becoming negative, *i.e.*, overload resources. Due to its diminishing returns, concavity promotes a "proactive" and robust allocation that will be possibly able to accommodate sudden or unexpected request load increase. Lipshitz-continuity implies that a small change of unutilized resources implies a bounded change in utility. Moreover, we additionally assume:

Assumption 2. *The utility $g(D)$ is $g(D) > 0$ iff $D > 0$, $g(D) = 0$ iff $D = 0$, and $g(D) < 0$ iff $D < 0$.*

Although Assumption 2 is not essential, it adds an intuitive perspective to the utility: Assuming negative utilities for negative residuals penalizes resource overloads.

Utility function formulation. Let the utility function over each residual resource $D \in \{D_{Mu}(\cdot), D_{uc}(\cdot), D_{ue}(\cdot)\}$ be defined as follows:

$$g(D) := \begin{cases} \log(D+1), & D > 0 \\ -|D|, & D \le 0 \end{cases}. \quad (3)$$

This function satisfies the properties of Assumptions 1 and 2, it is logarithmic for positive residuals and negative linear for negative ones, which implies a high penalization when the allocation results in insufficient reserved resources compared to those needed.

For notation simplicity, let

$$g_{uc}(t) := g\big(D_{uc}\big(\lambda_u(t), x_u(t)\big)\big),$$
$$g_{Mu}(t) := g\big(D_{uc}\big(\lambda_u(t), x_u(t), y_u(t)\big)\big),$$
$$g_{ue}(t) := g\big(D_{ue}\big(\lambda_u(t), x_u(t)\big)\big),$$

where $g(\cdot)$ as in (3) is the same across users and resources, and indices in $g_{ue}(t)$, $g_{Mu}(t)$ and $g_{uc}(t)$ denote the user and resource whose residuals are considered.

System utility function. For each user $u \in \mathcal{U}$ and resource type $r \in \{c, M, e\}$, *i.e.*, local and remote computation capacity and energy, let w_{ur} be an input weight over the resources' residual amount, that captures the importance of each user's devices w.r.t. resources. For example, a user whose device has limited energy autonomy may choose higher weight w_{ue} for energy compared to weights w_{uc} or w_{Mu} that she chooses for computing resources - local or remote. In this case, a higher weight w_{ue} for energy resources, which is this user's most limited resource, stimulates a higher utility for energy residuals compared the utility obtained for computation residuals. In practice weights can be requested directly by users through an app they use. Without loss of generality, we assume that

$$w_{uc} + w_{Mu} + w_{ue} = 1,$$

i.e., each user's u resource weights are normalized.

Then, the system's utility function $G\big(\boldsymbol{\lambda}(t), \mathbf{x}(t), \mathbf{y}(t)\big)$ is a weighted sum over the users' utilities for each residual resource:

$$G\big(\boldsymbol{\lambda}(t), \mathbf{x}(t), \mathbf{y}(t)\big) := \sum_{u\in\mathcal{U}} \sum_{r\in\{c,M,e\}} w_{ur} \cdot g_{ur}(t). \quad (4)$$

The system utility function $G(\cdot)$ has the properties of Assumptions 1 and 2, since it is the sum of functions that have them.

IV. Joint computation offloading and scheduling

We now present and analyse our optimization problems.

A. Optimization problem and optimum solution for known demand

We define the Joint Computation Offloading and Scheduling Problem (JCOSP) under known computation demand $\boldsymbol{\lambda}$, as:

Problem 1 (Offline JCOSP - max System Utility)**.**

$$\max_{\mathbf{x},\mathbf{y}} \quad G(\boldsymbol{\lambda}, \mathbf{x}, \mathbf{y}) \quad (5)$$

$$s.t. \quad x_u \in [0,1], \forall u \in \mathcal{U} \tag{6}$$

$$\sum_{u \in \mathcal{U}} y_u = 1 \tag{7}$$

Problem 1 aims at maximizing the utility derived from residual computation and energy resources. Eqs. (6) and (7) capture the possible values of offloading and scheduling variables. Negative utilities for negative residuals capture overloading costs. We indirectly incorporate energy- and computing- related constraints in our formulation by penalizing resource overloads.

Structural properties of the problem and its solution. Since $G(\cdot)$ is the sum of functions that follow Assump. 1 and 2, it holds:

Proposition 1. *For all $u \in \mathcal{U}$, the system utility function $G(\cdot)$ is concave w.r.t. both x_u and y_u.*

Since $G(\cdot)$ is concave w.r.t. both sets of decision variables, function $-G(\cdot)$ is convex. Thus Problem 1 is structurally equivalent to a convex minimization problem under upper and lower bounds and simplex constraints on the decisions, and it can be solved using standard convex optimization techniques [13]. The computational complexity for optimally solving the offline JCOSP is minimal, since convex optimization is a computationally easy operation.

Although in practice the computation demand is not known when the decisions are taken, studying the offline problem gives some useful insights about the main structural properties that will characterize the online problem.

B. Optimization problem for unknown demand

Adversarial optimization. The computation demand in real systems is highly unpredictable [3], and real-life applications (*e.g.*, AR) necessitate the assumption of non-stationary process of arriving computation requests. We thus decide both offloading and scheduling policies for slot t by the end of slot $t-1$, assuming that the computation demand may change arbitrarily.

Regret. The performance of an online algorithm Alg is characterized by *regret*: the difference over the entire horizon T, between the performance of Alg, and that of the "static" offline benchmark [14]. The static policy knows the evolution of demand $\boldsymbol{\lambda}(t)$ over time in hindsight, but it is limited to take *only one decision* over the entire horizon. In this work, the regret measures the difference in the system utility $G(\cdot)$ when following an online scheme instead of the static one, where the static policy decides one offloading and one scheduling policy, $\mathbf{x}^*$ and $\mathbf{y}^*$, not necessarily unique, s.t.:

$$(\mathbf{x}^*, \mathbf{y}^*) = \arg\max_{\mathbf{x},\mathbf{y}} \sum\nolimits_{t=1}^{T} G\big(\boldsymbol{\lambda}(t), \mathbf{x}, \mathbf{y}\big), \tag{8}$$

while satisfying (6) and (7). Let an online algorithm take decisions $\mathbf{x}(t)$ and $\mathbf{y}(t)$ for slot t. Its regret equals:

$$Reg(T) := \sum\nolimits_{t=1}^{T} \left(G\big(\boldsymbol{\lambda}(t), \mathbf{x}^*, \mathbf{y}^*\big) - G\big(\boldsymbol{\lambda}(t), \mathbf{x}(t), \mathbf{y}(t)\big)\right). \tag{9}$$

Problem formulation and structural properties. The typical goal in online settings is to find a *sequence of policies* that minimize regret over T. For $\mathbf{x}^*$ and $\mathbf{y}^*$ as defined above, we aim at finding a sequence of offloading and scheduling policies, $\mathbf{x}(t)$ and $\mathbf{y}(t)$, to solve the optimization problem:

Problem 2 (Online JCOSP - min System Regret)**.**

$$\min_{\mathbf{x}(t),\mathbf{y}(t)} \sum_{t=1}^{T} \left(G\big(\boldsymbol{\lambda}(t), \mathbf{x}^*, \mathbf{y}^*\big) - G\big(\boldsymbol{\lambda}(t), \mathbf{x}(t), \mathbf{y}(t)\big)\right). \tag{10}$$

$$s.t. \quad x_u(t) \in [0,1], \forall u \in \mathcal{U}, t = 1, \ldots, T, \tag{11}$$

$$\sum_{u \in \mathcal{U}} y_u(t) = 1, t = 1, \ldots, T. \tag{12}$$

This optimization problem aims at the minimization of the system regret, while the constraints capture the simplices where the decisions are constrained to belong. As in the offline case, our formulation indirectly incorporates the limitations in the available energy and computation resource by considering negative utilities, *i.e.*, interpreted as costs, when resource overloads occur. Structurally, the second summand of our objective function, $-G\big(\boldsymbol{\lambda}(t), \mathbf{x}(t), \mathbf{y}(t)\big)$ is convex, because it is the multiplication of $G(\cdot)$, which from Proposition 1 is concave, with -1. The objective function is a convex function in our decisions, because only the second summand depends on them, and it is convex. Observe that the two terms in (10) cannot be separated, which would imply solving Prob. 1 independently for each slot, because $G\big(\boldsymbol{\lambda}(t), \mathbf{x}^*, \mathbf{y}^*\big)$ depends on the arbitrary changing demand $\boldsymbol{\lambda}(t)$ during *all* the slots t of the horizon T.

V. Our optimal online algorithm

We present our methodology to go from state-of-the-art to our online learning algorithm, and we finally prove our algorithm's no-regret against the static benchmark, and its scalability.

"No regret": asymptotically optimal online decisions. The performance of online algorithms is evaluated asymptotically over the horizon T, through their regret. If an algorithm's regret scales sublinearly with the horizon T [14], *i.e.*, if $Reg(T) = o(T)$, or, equivalently, $\lim_{T \to +\infty} \frac{Reg(T)}{T} = 0$, the algorithm has "no regret" against the benchmark. This means that it learns to perform asymptotically as well as the benchmark as $T \to +\infty$.

Challenges. In this work, the design of efficient algorithms is obstructed because: (a) Decisions are taken online and *without any assumption* on $\boldsymbol{\lambda}(t)$. (b) *Offloading and scheduling decisions are interrelated with each other*, which could result in the impact of offloading decisions being neutralized by scheduling decisions, and vice versa. To design our efficient online algorithm, we will thus rely only on the convexity of the utility function.

Online Mirror Descent (OMD). We build on this class of online schemes, that has no regret against the static benchmark [14]. We present its generic form in Algorithm 1. The main idea in OMD is to produce a new solution by updating an existing one. The update consists of two stages: (i) perform a step against the gradient, and (ii) project this point back to the feasible space, in order to ensure that the problem's constraints are satisfied.

In more detail, OMD takes as input a regularization function $R(\cdot)$ that "stabilizes" the online solutions $\mathbf{a}(t)$ and must be strongly-convex w.r.t. a norm [15]. OMD uses an auxiliary variable $\mathbf{b}(t)$ of dimensions equal to those of the solution $\mathbf{a}(t)$, which initializes as in (14) and updates as in (15), taking a step against the gradient of the objective. Observe that the initialization of both the decision vector $\mathbf{a}(1)$ and the auxiliary vector $\mathbf{b}(1)$ is

Algorithm 1 Online Mirror Descent (OMD) [14]

Input: Regularization function $R(\cdot)$, stepsize η, feasible set $\mathcal{A}$.
Output: Decision $\mathbf{a}(t), \forall t = 1...T$

1: Initialize auxiliary variable $\mathbf{b}(1)$:
$$\nabla R\big(\mathbf{b}(1)\big) = \mathbf{0} \tag{14}$$
2: Initialize solution: $\mathbf{a}(1) = \arg\min_{\mathbf{a}\in\mathcal{A}} \big\{B_R\big(\mathbf{a}, \mathbf{b}(1)\big)\big\}$
3: **for** $t = 1, 2, ..., T$ **do**
4: Evaluate decision $\mathbf{a}(t)$ on the objective $G\big(\mathbf{a}(t)\big)$
5: Update auxiliary vector $\mathbf{b}(t)$ so that:
$$\nabla R\big(\mathbf{b}(t+1)\big) = \nabla R\big(\mathbf{b}(t)\big) - \eta \nabla G\big(\mathbf{a}(t)\big) \tag{15}$$
6: Project to obtain feasible decision $\mathbf{a}(t+1)$:
$$\mathbf{a}(t+1) = \arg\min_{\mathbf{a}\in\mathcal{A}} \big\{B_R\big(\mathbf{a}, \mathbf{b}(t+1)\big)\big\} \tag{16}$$
7: **end for**

not related to the solution of the offline instance of the problem, but it only depends on the regularization function $R(\cdot)$. In each slot t, OMD evaluates the online decisions $\mathbf{a}(t)$ by observing its utilities $G(\mathbf{a}(t))$. It then updates $\mathbf{b}(t)$ and projects it to the feasible space to obtain a solution $\mathbf{a}(t+1)$. The projection in (16) minimizes $\mathbf{b}(t)$'s Bregman divergence $B_R(\mathbf{a}, \mathbf{b})$ w.r.t. the regularization function, where $B_R(\mathbf{a}, \mathbf{b})$ is defined as:

$$B_R(\mathbf{a}, \mathbf{b}) := R(\mathbf{a}) - R(\mathbf{b}) - \langle \nabla R(\mathbf{b}), \mathbf{a} - \mathbf{b} \rangle . \tag{13}$$

A carefully chosen regularization $R(\cdot)$ can eliminate the projection of step 6 in Alg. 1. Eliminating step 6, while keeping the remaining steps same, decreases the solution's computational complexity.

Regularization. A common regularization function for simplex feasible spaces is the negative entropy

$$r(\mathbf{a}) = \sum_{j=1}^{n} a_j \log a_j, \ \forall \mathbf{a} \in \mathbb{R}^n. \tag{17}$$

From (11) and (12), both the computation offloading decisions x_u regarding each user u and the server's scheduling decisions $\mathbf{y}$, form simplices. We will adapt $r(\cdot)$, and thus OMD, to our setting in order to jointly regularize scheduling and offloading decisions. We define our *modified* entropic function $R(\mathbf{x}, \mathbf{y})$ as follows:

$$R(\mathbf{x}, \mathbf{y}) := r(\mathbf{y}) + \sum_{u\in\mathcal{U}} r(\mathbf{x}_u), \tag{18}$$

where $\mathbf{x}_u(t) := \big(x_u(t), 1 - x_u(t)\big)$ and $r(\cdot)$ is defined in (17).

Proposition 2. *$R(\mathbf{x}, \mathbf{y})$ is $\big(|\mathcal{U}| + 1\big)$-strongly convex.*

Sketch of proof. $r(\cdot)$ in (17) is 1-strongly convex w.r.t. the 1-norm [15], *i.e.*, $r(\mathbf{a}_1) \geq r(\mathbf{a}_2) + \langle \nabla r(\mathbf{a}_1), \mathbf{a}_2 - \mathbf{a}_1 \rangle + \frac{1}{2}\|\mathbf{a}_1 - \mathbf{a}_2\|_1$. We sum one such inequality for $\mathbf{y}$ and one for each x_u. Eq. (18) holds due to interchangeability of the sum and the dot-product. □

OJOSO: Optimal Joint Online Scheduling & Offloading algorithm. Let $\mathbf{b}^{\mathbf{x}} = \{(b^{\mathbf{x}}_{u1}, b^{\mathbf{x}}_{u2})\}_u$ and $\mathbf{b}^{\mathbf{y}} = \{b^{\mathbf{y}}_u\}_u$ be the auxiliary vectors of dimension $|\mathcal{U}| \times 2$ and $|\mathcal{U}| \times 1$ for offloading and scheduling decisions, respectively. Vectors $\mathbf{b}^{\mathbf{x}}$ and $\mathbf{b}^{\mathbf{y}}$ are the adaptation of the auxiliary vector $\mathbf{b}$ of OMD. Applying (14) over $R(\cdot)$ in (18) to initialize the auxiliary matrices we get

$$b^{\mathbf{x}}_{u1}(1) = b^{\mathbf{x}}_{u2}(1) = 1/2 \quad \text{and} \quad b^{\mathbf{y}}_u(1) = 1/|\mathcal{U}|, \forall u \in \mathcal{U},$$

Algorithm 2 Optimal Joint Online Scheduling and Offloading

Input: Regularization function $R(\cdot)$, stepsize η, system utility $G(\cdot)$.
Output: Offloading policy $\mathbf{x}(t)$, scheduling policy $\mathbf{y}(t)$, $\forall t = 1...T$

1: Initialize offloading and scheduling decisions by equally distributing the users computation demand for remote and local execution, and equally splitting the server's computation capacity among users:
$$x_u(1) = \frac{1}{2}, \quad \text{and} \quad y_u(1) = \frac{1}{|\mathcal{U}|}, \qquad \forall u \in \mathcal{U}. \tag{23}$$
2: **for** $t = 2, ..., T$ **do**
3: Apply $\mathbf{x}(t)$ and $\mathbf{y}(t)$ and observe utility $G\big(\boldsymbol{\lambda}(t), \mathbf{x}(t), \mathbf{y}(t)\big)$
4: For $\nabla G(t)|_z$, $z \in \{x_u, (1 - x_u), y_u\}$ as in (19),
Update offloading by incorporating gradient feedback and renormalizing decisions *per user*:
$$x_u(t+1) = \frac{x_u(t) e^{-\eta \nabla G(t)|_{x_u}}}{A}, \text{where} \tag{24}$$
$$A = x_u(t) e^{-\eta \nabla G(t)|_{x_u}} + \big(1 - x_u(t)\big) e^{\nabla G(t)|_{(1-x_u)}}$$
Update Scheduling by incorporating gradient feedback and renormalizing *over all users*:
$$y_u(t+1) = \frac{y_u(t) e^{-\eta \nabla G(t)|_{y_u}}}{\sum_{u\in\mathcal{U}} y_u(t) e^{-\eta \nabla G(t)|_{y_u}}} \tag{25}$$
5: **end for**

which belong in the feasible space. Thus, without projection, we get the initialization of line 1 in Algorithm 2. The initialization of both $b^{\mathbf{x}}_{u1}(1)$ and $b^{\mathbf{y}}_u(1)$ for $u \in \mathcal{U}$ does not rely on knowledge of computation demand at slot t. In fact, this initialization depends only on our modified regularization function $R(\cdot)$ of (18).

To update $\mathbf{b}^{\mathbf{x}}(t)$ and $\mathbf{b}^{\mathbf{y}}(t)$, we apply (15) with $R(\cdot)$ as in (18). For notation simplicity, let

$$\nabla G(t)|_z := \frac{\partial G\big(\boldsymbol{\lambda}(t), \mathbf{x}(t), \mathbf{y}(t)\big)}{\partial z}. \tag{19}$$

Then, after algebraic manipulations, the elements $b^{\mathbf{x}}_{u1}(t), b^{\mathbf{x}}_{u2}(t)$ and $b^{\mathbf{y}}_u(t)$, of the auxiliary vectors $b^{\mathbf{x}}(t)$ and $b^{\mathbf{y}}(t)$, respectively, are updated following the rule:

$$b(t+1) = b(t) \exp\big\{-\eta \nabla G(t)|_b\big\}, \tag{20}$$

where $b(\cdot)$ any of the $b^{\mathbf{x}}_{u1}(\cdot), b^{\mathbf{x}}_{u2}(\cdot)$ and $b^{\mathbf{y}}_u(\cdot)$. We project to the feasible space by combining (13), (18) and (20). To obtain the updates in (24) and (25), we differentiate and get the minimizers

$$\mathbf{x}(t+1) = \mathbf{b}^{\mathbf{x}}(t+1), \tag{21}$$
$$\mathbf{y}(t+1) = \mathbf{b}^{\mathbf{y}}(t+1). \tag{22}$$

We ensure feasibility by normalizing (21) and (22), and we conclude to the updates in (24) and (25).

In the Appendix we prove that for OJOSO's regret it holds:

Theorem 1 (OJOSO's No-regret property). *For a convex objective function $G(\cdot)$, which is Lipschitz-continuous with constant L, a stepsize η and a time horizon T, OJOSO's regret is bounded by:*

$$Reg(T) \leq \frac{\log|\mathcal{U}| + |\mathcal{U}|}{\eta} + \frac{\eta L^2 T}{2(|\mathcal{U}| + 1)}, \tag{26}$$

Additionally, for stepsize $\eta = \sqrt{\frac{2(\log|\mathcal{U}| + |\mathcal{U}|)(|\mathcal{U}|+1)}{L^2 T}}$, we

get:

$$Reg(T) \leq L\sqrt{T}\sqrt{\frac{2\left(\log|\mathcal{U}|+|\mathcal{U}|\right)}{(|\mathcal{U}|+1)}} \leq 2L\sqrt{T}. \quad (27)$$

Regret scaling properties. From (27), OJOSO's *regret scales sublinearly to the horizon* T, *i.e.*, it performs on average as well as the static benchmark. OJOSO inherits the $\sqrt{T}$ dependence on T from OMD, from where it stems. OMD's regret is logarithmic on the dimension of the problem (Corol. 2.16 in [15]), which in our case equals $|\mathcal{U}+1|$, *i.e.*, the users' devices and the server. However, our modified regularization in (18) leads to a regret that is *independent of the number* $|\mathcal{U}|$ *of users* in the system. OJOSO is thus scalable, and it learns good joint offloading and scheduling policies regardless of the number of devices in the system.

OJOSO in action. OJOSO is executed at the edge server, in computing resources reserved by the agent that takes decisions, *e.g.*, the provider of the application which generates the computing tasks of the users. The sequence of steps and exchanges between the edge server and the users are:

Initialization, before the beginning of slot $t = 1$: (i) The users inform the agent about the available resources at their devices: They send their processing speed f_u and energy availability e_u to the agent. (ii) The agent initializes the offloading and scheduling decisions, $\mathbf{x}(1)$ and $\mathbf{y}(1)$, for the first slot, as in (23). (iii) The agent sends $\mathbf{x}(1)$ to the users.

For each slot $t = 1, \ldots, T$: (i) *Computation demand* $\boldsymbol{\lambda}(t)$ *is revealed and decisions* $\mathbf{x}(t)$ *and* $\mathbf{y}(t)$ *that were computed at the previous slot by the agent are applied by both users and the server:* Users send a packet to the agent, containing the number $\boldsymbol{\lambda}(t)$ of computing tasks that they generated. They apply over $\boldsymbol{\lambda}(t)$ the offloading decision $\mathbf{x}(t)$, that the agent sent them during slot $(t-1)$, *i.e.*, user u offloads $\lambda_u(t)x_u(t)$ tasks for remote execution at the server. The server schedules the offloaded tasks based on the scheduling policy $\mathbf{y}(t)$. (ii) *Decisions* $\mathbf{x}(t)$ *and* $\mathbf{y}(t)$ *are evaluated:* The agent computes the objective gradient $\nabla G\big(\boldsymbol{\lambda}(t), \mathbf{x}(t), \mathbf{y}(t)\big)$ by substituting the values of $\boldsymbol{\lambda}(t)$, $\mathbf{x}(t)$ and $\mathbf{y}(t)$ to its formula. (iii) *Decisions* $\mathbf{x}(t)$ *and* $\mathbf{y}(t)$ *are updated by the agent to obtain* $\mathbf{x}(t+1)$ *and* $\mathbf{y}(t+1)$*:* The agent uses formulas (24) and (25) to update decisions and sends to users offloading decisions $\mathbf{x}(t+1)$.

Computational complexity and memory overhead. OJOSO is an extremely fast and lightweight algorithm. In fact, both its initialization step and all the actions within all time slots are straightforward and have a fixed cost $O(1)$ each. Indeed, they only need to either send a packet or to perform a predefined assignment. Moreover, there is no memory overhead: Once the value of $\boldsymbol{\lambda}(t)$ is used during t, it is not useful anymore, and can be discarded.

VI. Numerical Evaluation

Setup. We consider a horizon of $T = 1000$ slots and tasks with normalized computation load and input data, *i.e.*, $l = d = 1$. We generate computation demand $\lambda_u(t)$ in the normalized interval $[0, 100]$ through three processes: (a) Uniformly random: $\lambda_u(t) \sim U[1, 100]$, (b) Sinusoid adversarial: $\lambda_u(t) = 50 + 40\sin(t\pi/12) + n_t$, where $n_t \sim U[-10, 10]$ a uniformly distributed noise, and

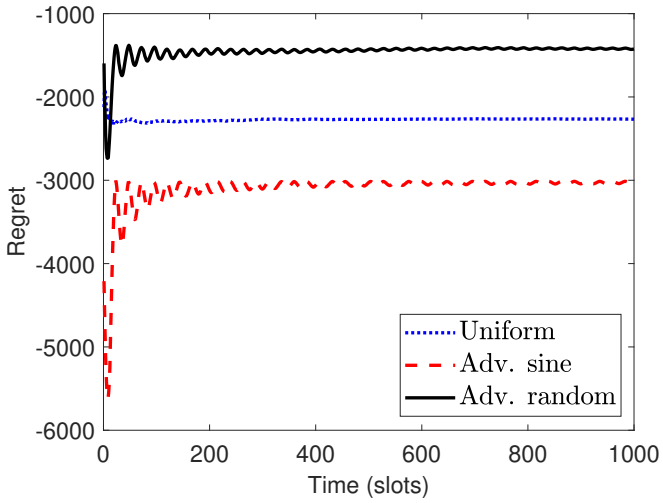

Figure 1: OJOSO's regret w.r.t. static benchmark

(c) Random adversarial: $\lambda_u(t) = 50 + X_t + n_t$, where $X \sim [-40\sin(t\pi/12), 40\sin(t\pi/12)]$ and n_t as before. Distribution (a) is stationary, but (b) and (c) are non-stationary, because their mean value and variance change with time. Generating the computation demand vectors $\boldsymbol{\lambda}(t)$ through the above procedures obstructs the operation of online algorithms, that apply decisions $\mathbf{x}(t)$ and $\mathbf{y}(t)$ over the computation demand $\boldsymbol{\lambda}(t-1)$ of the most recent past. Indeed, for $\boldsymbol{\lambda}$ generated through (a), the computation demands $\boldsymbol{\lambda}(t)$ and $\boldsymbol{\lambda}(t-1)$ will vary on average 50 units, while for $\boldsymbol{\lambda}$ generated through (b) and (c) the intervals from which $\boldsymbol{\lambda}(t)$ and $\boldsymbol{\lambda}(t-1)$ are drawn additionally change over time.

We let $|\mathcal{U}| = 100$ users with two levels of normalized computing and energy capacity: $f_u \in \{F, 2F\}$, $e_u \in \{E, 2E\}$, *i.e.*, four user categories w.r.t their available resources: High-energy-High-speed (HeHs), High-energy-Low-speed (HeLs), Low-energy-High-speed (LeHs), and Low-energy-Low-speed (LeLs). Under any offloading policy, LeHs users have enough resources to accommodate at most half of their maximum demand, HeHs and LeLs can accommodate it exactly, and HeLs could accommodate the double of it. The server can accommodate all demand.

Our utility functions $g(\cdot)$, as presented in (3), are logarithmic for non-negative residuals and linear for negative, severely penalizing possible resource overloads that may occur. Let x_u^* and y_u^* be the offloading and scheduling decisions of the static for user u, and $x_u(t)$ and $y_u(t)$ be OJOSO's respective decisions.

In Fig. 1 we depict OJOSO's regret for the three presented synthetic datasets. We can make two main observations:

- **OJOSO is a no-regret algorithm.** We confirm our theoretical result of Theorem 1, since its regret scales sublinearly with the time horizon in all datasets (i)-(iii).
- **OJOSO adapts to unpredictable demand changes.** The regret starts from low negative values, and it is stabilized in higher, but always negative, values, Thus, it performs on average better than the static policy, obtaining a utility higher than the static. Intuitively, while the static is constrained to take only one decision, OJOSO dynamically adapts to the demand unpredictability, learning to optimally offload and schedule, despite deciding under lack of information about

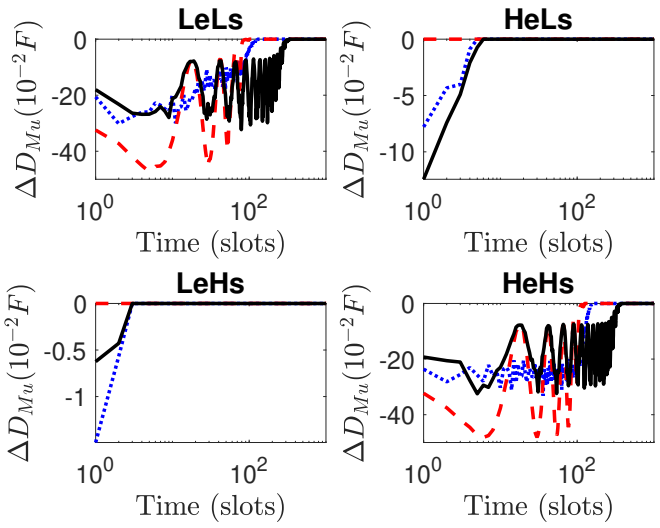

Figure 2: ΔD_{Mu} *vs.* time,*vs.* user category, for time horizon $T = 1000$ (x-axes in log scale), for three adversarial datasets: "uniform" in blue points, "adversarial sine" in red dashes, and "adversarial random" in black solid line.

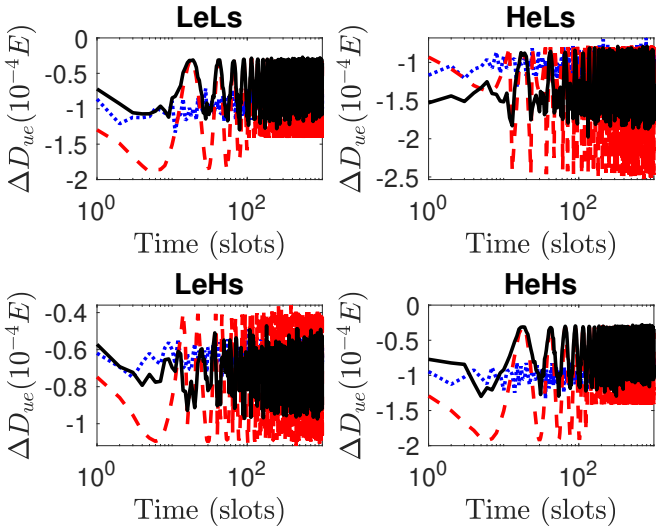

Figure 3: ΔD_{ue} *vs.* time,*vs.* user category, for time horizon $T = 1000$ (x-axes in log scale), for three adversarial datasets: "uniform" in blue points, "adversarial sine" in red dashes, and "adversarial random" in black solid line.

the computation demand over which they will be applied.

We now depict the difference between the residuals (without applying a utility over them) obtained by the static and those obtained by OJOSO, *i.e.*,

$$\Delta D_{ue}(t) = \sum_{u\in\mathcal{U}} \left(D_{ue}\left(\lambda_u(t), x_u^*\right) - D_{ue}\left(\lambda_u(t), x_u(t)\right)\right),$$

$$\Delta D_{Mu} = \sum_{u\in\mathcal{U}} \left(D_{Mu}\left(\lambda_u(t), \pi_u^*, w_u^*\right) - D_{ue}\left(\lambda_u(t), \pi_u(t), w_u(t)\right)\right).$$

In Fig. 2 we depict ΔD_{Mu} per user category, focusing on the server's computing residual resources. We observe that:

- **OJOSO learns the optimal amount of computations to offload.** Indeed, $\Delta D_{uc} \to 0$ for all user categories. All subplots here start from negative values and converge to zero, which indicates that at the beginning of the time horizon OJOSO performs better than the static and that it later learns how many tasks to offload for remote execution.
- **OJOSO learns to share the server's resources.** For HeLs and LeHs users, OJOSO's scheduling learns extremely fast ($t = 10$) to behave as that of the static: it allocates more computing resources to LeHs, letting them offload more tasks for remote allocations, while it locally executes HeLs's demand, given their increased capabilities. The opposite holds for HeHs, who have enough resources for executing all of their demand locally, for which OJOSO needs more time ($t = 130$) to learn the resources to allocate.

In Fig. 3 we depict ΔD_{ue} per user category, focusing at the residual energy resources at the users' devices. We observe that:

- **OJOSO learns to identify and respect energy-limited devices.** Although ΔD_{ue} oscillate, they are always negative, *i.e.*, the energy residuals left by OJOSO are higher that those left by the static, *i.e.*, all user devices use less energy under OJOSO compared to what they use under the static policy.

Similar results holds also for computation residuals locally, *i.e.*,

$$\Delta D_{uc}(t) = \sum_{u\in\mathcal{U}} \left(D_{uc}\left(\lambda_u(t), x_u^*\right) - D_{uc}\left(\lambda_u(t), x_u(t)\right)\right),$$

but due to space limitations we will omit them. Our main observation for these residuals are that: (i) OJOSO learns the optimal amount of computations to execute locally, and (ii) OJOSO learns to identify and respect computationally-limited devices.

VII. Conclusions

We consider a multi-user MEC system with limited energy autonomy for the mobile devices and with limited computing capabilities for both mobile devices and the edge server where users can offload part of their computation load. We consider a utility over "resource residuals", that capture the difference between the assigned and the needed resources, and we aim at the regret minimization, *i.e.*, of the difference between the utility obtained by the static offline benchmark that knows the system evolution in hindsight, and our online decisions. We design an algorithm that jointly learns policies for offloading computations and scheduling them for execution at the shared MEC server. We prove that it is asymptotically optimal, *i.e.*, it has no regret, and that its performance is independent of the number of devices in the system. From our numerical evaluation, we conclude that OJOSO adapts to unpredictable demand changes, it learns to identify resource-limited devices, and it learns to share the server's compute resources so as to optimize the defined utility function.

VIII. Acknowledgement

This paper was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "1st Call for H.F.R.I. Research Projects to support Faculty Members & Researchers and the Procurement of high-cost research equipment grant" (Project Number: HFRI-FM17-352, Project Title: Wireless Mobile Delay-Tolerant Network Analysis and Experimentation, Project acronym: LEMONADE).

APPENDIX

Proof of Theorem 1. Let $\mathbf{w} := (\mathbf{x}, \mathbf{y})$ be a $(|\mathcal{U}|+1)$x1 array that captures the vertical concatenation of the decisions $\mathbf{x}$ and $\mathbf{y}$. Let $\mathbf{w}^* := (\mathbf{x}^*, \mathbf{y}^*)$ be an optimal static decision over T time slots as in (8). For notation simplicity, let

$$\mathbf{z}(t) := \nabla G\big(\boldsymbol{\lambda}(t), \mathbf{x}(t), \mathbf{y}(t)\big),$$
$$G(\mathbf{w}(t)) := G\big(\boldsymbol{\lambda}(t), \mathbf{x}(t), \mathbf{y}(t)\big),$$
$$B = (|\mathcal{U}|+1).$$

From Theorem 2.21 in [15] we get

$$\sum_{t=1}^{T} \langle \mathbf{z}(t), \mathbf{w}(t) - \mathbf{w}^* \rangle \leq \frac{R(\mathbf{w}^*) - R\big(\mathbf{w}(1)\big)}{\eta} + \sum_{t=1}^{T} \frac{\eta \|\mathbf{z}(t)\|_q^2}{2B}, \tag{28}$$

where the q-norm is the dual norm of the p-norm w.r.t. which the regularization $R(\cdot)$ is B-strongly-convex. Our utility $G(\cdot)$ is Lipschitz-continuous, thus it exists a positive constant L such that:

$$L \geq \|\mathbf{z}(t)\|_q, \tag{29}$$

for all q-norms. Then, due to eq. (29), eq. (28) becomes:

$$\sum_{t=1}^{T} \langle \mathbf{z}(t), \mathbf{w}(t) - \mathbf{w}^* \rangle \leq \frac{R(\mathbf{w}^*) - R\big(\mathbf{w}(1)\big)}{\eta} + \sum_{t=1}^{T} \frac{\eta L^2}{2B}, \tag{30}$$

All decision variables belong in $[0, 1]$, which implies that $r(\mathbf{x}_u) < 0$ and $r(\mathbf{y}) < 0$. Combining with (18), trivially:

$$R(\mathbf{w}^*) \leq 0. \tag{31}$$

Based on OJOSO's initialization, it is:

$$R\big(\mathbf{w}(1)\big) \overset{(18)}{=} -\log|\mathcal{U}| - |\mathcal{U}|\log 2 = -\log|\mathcal{U}| - |\mathcal{U}|. \tag{32}$$

Combining (30), (31) and (32) we get:

$$\sum_{t=1}^{T} \langle \mathbf{z}(t), \mathbf{w}(t) - \mathbf{w}^* \rangle \leq \frac{\log|\mathcal{U}| + |\mathcal{U}|}{\eta} + \frac{\eta L^2 T}{2B}, \tag{33}$$

where we omitted the first term in the Right Hand Side (RHS) of (30) because it is negative. Due to convexity of $G(\cdot)$ it holds:

$$G\big(\mathbf{w}(t)\big) - G(\mathbf{w}^*) \leq \langle \mathbf{z}(t), \mathbf{w}(t) - \mathbf{w}^* \rangle. \tag{34}$$

Then:

$$Reg(T) \overset{(9),(8)}{\leq} \sum_{t=1}^{T} \big(G\big(\mathbf{w}(t)\big) - G(\mathbf{w}^*)\big) \overset{(34)}{\leq} \sum_{t=1}^{T} \langle \mathbf{z}(t), \mathbf{w}(t) - \mathbf{w}^* \rangle.$$

The first inequality holds by definition and the second is due to the convexity of $G(\cdot)$. Substituting the RHS of (33) concludes the first part of the proof. For the second part, it is easily verifiable that (26) is minimized for η as given in Theorem.

REFERENCES

[1] W. Zhang, B. Han, and P. Hui, "On the networking challenges of mobile augmented reality," in *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, VR/AR Network '17, pp. 24–29, ACM, 2017.

[2] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Com. Surveys Tutor.*, vol. 19, no. 4, pp. 2322–2358, 2017.

[3] N. Liakopoulos, G. Paschos, and T. Spyropoulos, "No regret in cloud resources reservation with violation guarantees," in *proc. IEEE International Conference on Computer Communications - INFOCOM*, pp. 1747–1755, 2019.

[4] S. Mahmoodi, R.Uma, and K.Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Trans. on Cloud Computing, Special Issue on Mobile Clouds, Vol. 7, No. 2, pp. 301-313*, 2016.

[5] Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, "Partial offloading scheduling and power allocation for mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6774–6785, 2019.

[6] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 17, no. 3, pp. 1784–1797, 2018.

[7] LE. Chatzieleftheriou, A. Destounis, GP. Paschos and I. Koutsopoulos, "Blind optimal user association in small-cell networks," in *proc. IEEE International Conference on Computer Communications - INFOCOM*, 2021.

[8] T. Chen, Y. Shen, Q. Ling, and G. B. Giannakis, "Online learning for "thing-adaptive" fog computing in IoT," in *Asilomar Conf. on Signals, Systems, and Computers*, pp. 664–668, 2017.

[9] M. Shoaib, S. Venkataramani, X. Hua, J. Liu, and J. Li, *Exploiting On-Device Image Classification for Energy Efficiency in Ambient-Aware Systems*, ch. Mobile Cloud Visual Media Computing, pp. 167–199. Springer, 2015.

[10] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*. Elsevier, 2012.

[11] S. Mannor, J. N. Tsitsiklis, and J. Y.Yu, "Online learning with sample path constraints," *Journal of Machine Learning Research*, vol. 10, pp. 569–590, 2009.

[12] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Trans. on Networking*, vol. 1, pp. 344–357, June 1993.

[13] D. Bertsekas, *Convex Optimization Algorithms*. Athena Scientific, 2015.

[14] E. Hazan, "Introduction to Online Convex Optimization," *Foundations and Trends in Optimization*, vol. 2, no. 3-4, pp. 157–325, 2015.

[15] S. Shalev-Shwartz, "Online Learning and Online Convex Optimization," *Found. and Trends Machine Learning*, vol. 4, no. 2, pp. 107–194, 2012.