Digital Object Identifier

A Stackelberg Game approach for Managing AI Sensing Tasks in Mobile Crowdsensing

HAMTA SEDGHANI[1,3], MINA ZOLFY LIGHVAN[1], HADI S. AGHDASI[1], MAURO PASSACANTANDO[2], GIACOMO VERTICALE[3](MEMBER, IEEE), AND DANILO ARDAGNA[3](MEMBER, IEEE)

[1]Faculty of Electrical and Computer Engineering, University of Tabriz, Iran
[2]Department of Computer Science, University of Pisa, Pisa, Italy (e-mail: mauro.passacantando@unipi.it)
[3]Department of Electronics, Information and Bioengineering, Politecnico di Milano, Milan, Italy (e-mail: name.lastname@polimi.it)

Corresponding author: Hamta Sedghani (h.sedghani@tabrizu.ac.ir, hamta.sedghani@polimi.it).

The European Commission has partially funded this work under the H2020 grant n. 101016577 AI-SPRINT: AI in Secure Privacy pReserving computINg conTinuum.
This paper has been published in IEEE Access, vol. 10, pp. 91524-91544, see:
https://ieeexplore.ieee.org/document/9866032.

ABSTRACT Mobile Crowdsensing (MCS) is a new paradigm that leverages the collective sensing ability of a crowd so that a special task can be performed through the aggregation of information collected from personal mobile devices. While MCS brings several benefits, its application is prevented by challenges such as the efficient recruitment of users, effective mechanisms for rewarding users to encourage participation, and an effective and fast enough approach for managing the underlying resources that support large-scale MCS applications involving a large number of people in data collection. On the other hand, Artificial Intelligence (AI) applications, which are mostly based on Deep Neural Networks (DNN), are becoming pervasive today and are executed by the end users' mobile devices, which are characterised by limited memory and computing power, and low battery level.

This paper describes and evaluates an incentive mechanism for a mobile crowdsensing system with an AI sensing task based on a one-leader multi-follower Stackelberg game. The MCS platform, as a leader, provides an AI sensing task to be executed by a DNN, which can be deployed in two different ways: fully on the user device or partially on the device and partially on edge or cloud resources. The users, as followers, make their decisions regarding their participation to the MCS system and select their desired deployment given the energy and memory available on their device and the deployment reward proposed by the MCS platform. The goals of the MCS platform are: i) to motivate the users to participate in the system, ii) to maximize its profit, and iii) to identify the optimal resources supporting the sensing task that minimizes the cost and provide performance guarantees.

This problem has been formulated as a mixed integer nonlinear program and propose an efficient algorithmic approach to solve it quickly. The proposed approach has been compared with some baseline methods and with BARON state-of-the-art solver. Results show that our approach converges to the optimal solution much faster than BARON (up to orders of magnitude) especially in large scale systems. Furthermore, the comparison to the baseline methods shows that our approach always beats the best baseline method under different scenarios providing up to 16% improvement for the MCS platform profit.

INDEX TERMS Mobile Crowdsensing, AI sensing tasks, Incentive mechanism, Stackelberg game.

I. INTRODUCTION

Mobile Crowdsensing (MCS) is a distributed paradigm to use the sensing and computing capabilities of end-user's devices for environmental data collection and analysis. MCS systems are generally used in two different scenarios based on the type of user involvement [1]: i) *Participatory* [2] where the users participate in the system voluntarily and the user manual intervention is needed for certain input; ii)

Opportunistic [3] where the data is gathered automatically in the background without user intervention.

Artificial Intelligence (AI) applications and Deep Learning (DL) have been growing rapidly in these years and cloud computing and its technologies have improved AI very much [4]. The combination of AI and cloud computing results in an extensive network capable of holding massive volumes of data while continuously learning and improving. Moreover, nowadays AI applications have been moving toward mobile computing and Internet of Things (IoT) [5] while cloud resources are usually far from the mobile devices and this long distance might be the cause of long latency. However, end user's devices may not be powerful enough to process some heavy tasks such as the execution of AI or DL applications. To solve this issue, edge computing establishes a new computing paradigm that moves AI and machine learning close to where the data generation and computation actually take place.

Nowadays, due to the pervasiveness of AI, a significant number of real world applications are based on neural networks or deep neural networks (DNN), which include a large number of layers. One clear example comes from Augmented Reality (AR) applications [6]. AI technologies such as machine learning and deep learning are well suited to AR applications because they can collect data by a camera and also integrate some other data from the device's gyroscopes, sensors, accelerometers and GPS. For example, in recent navigation systems, novel AI applications using cloud processing help to add scene descriptions and to integrate the data coming from the gyroscopes and GPS positions to provide a safe route.

The resource shortage of mobile devices in MCS systems could be more sensitive when the applications have quality of service (QoS) response time constraints. Hence, edge paradigm allows faster computing and it can enhance the performance of AI-enabled applications and keep the operating costs down. Partitioning the DNN of AI application component and running partially on local device and assigning the other partitions to the existing resources on edge or cloud is a common approach to use optimally the resources on computing continua while satisfying QoS constraints [7], [8].

The limited capacity of mobile devices to execute the sensing tasks with heavy process on one hand and the cost of running the application on edge or cloud on the other hand make the resource allocation problem across the computing continuum very difficult. Game theory models are needed to support the AI application execution taking into account: i) on the field the memory capacity and energy available at the mobile devices and MCS platform's budget for pricing the tasks, and ii) on the edge/cloud the costs to guarantee a given latency threshold and the competition to access the resources at the MCS platform site (e.g., a set of small datacenters located inside the 5G Radio Access Network (RAN)) and in the remote cloud. There are a lot of existing works [9]–[17] focusing on DNN partitioning of AI application components which are run by distributing computations among mobile devices and edge or cloud. On the other hand, many works focus on incentivizing the smart devices users to run different sensing tasks in MCS by relying on auction mechanism [18]–[24], while some others rely on game theory [25]–[32]. However, none of recent works considered MCS systems based on AI applications where the computational load needs to be dynamically partitioned across the computing continuum and the smartphone resources of all the users involved. Previous literature assumes light tasks that all smart devices are able to run, whereas current MCS applications are also characterized by heavy tasks, like image processing, and their adoption is steadily increasing [33]. To the best of our knowledge, this is the first work considering heavy tasks in MCS systems which tries to balance the computing load of the task between the user's local device and edge/cloud resources. As the first literature work, this paper: i) considers a MCS system with the goal of incentivizing the users to participate and run an AI sensing task with DNN running on their local device or edge/cloud; ii) allocates tasks according to the memory and energy available at the users' local device while guaranteeing a response time performance constraint. This problem is formulated as a bi-level game and the results show that the proposed approach converges to the optimal solution much faster than the BARON state-of-the-art solver.

In this work, MCS platform is located in the edge layer (e.g., in one or multiple 5G towers) and some mobile augmented reality (AR) sensing tasks are run by users through their devices. The computation required by AI sensing tasks can also be offloaded to the cloud when edge servers are saturated. The interaction among the MCS platform and the end users is formulated as a Stackelberg game aiming at: i) maximizing the MSC provider utility, ii) optimizing the edge and cloud resource management while iii) guaranteeing users' device memory and energy constraints, and iv) fulfilling application performance constraints.

The aim is to motivate the users to run the AI sensing tasks: each user considers running the task as a cost (based on the device energy consumption) and needs to receive a specific price from the MCS platform as a reward. The users are motivated to run the task only if the cost of running the task is less than their reward.

In summary, the main contributions of this paper are the following:

1) The interaction among multiple mobile users and MCS platform is formulated as a one-leader multi-follower Stackelberg game, where the platform is the leader and the users are the followers.
2) An approach based on KKT conditions is developed to find the optimal number of edge and cloud resources and to solve the mixed integer nonlinear problem underlying the Stackelberg game quickly.
3) The performance of the proposed approach is finally evaluated by comparing our solution algorithm with the BARON state-of-the-art solver. Experimental results

show that this approach can find the optimal solution much faster than BARON.

The remainder of this paper is organized as follows. Related works are discussed in Section II. Section III describes the mobile edge cloud system model, Section IV formulates the platform and user problems, while the Stackelberg game model is described in Section V. In Section VI, the heuristic solution approach is explained. Experimental results are discussed in Section VII, while conclusions are finally drawn in Section VIII.

II. RELATED WORK

The growth of interest in IoT devices, the data those devices produce and the ability of DL based solutions to generate valuable inferences from this massive volume of data, caused a growth of interest in MCS and resource management in AI applications. In this section, the recent works published in these two areas are investigated.

A. INCENTIVE MECHANISMS IN MCS

The most important issue in MCS systems is to motivate the smart device users to participate and perform the sensing tasks. Game theory methods, specially Stackelberg game and auction mechanism are the most common way to incentivize the users to run the sensing tasks and receive a reward in return. An incentive mechanism based on Stackelberg game has been proposed in [29], where the MCS platform estimates the users' utility function parameters and solves the optimization problem of the game in a centralized manner instead of exchanging a lot of messages in distributed methods.

In [34], authors proposed a three-step mechanism based on Stackelberg game to tackle three problems: pricing the incoming tasks by platform, selecting the favourite tasks by users and finally worker selection by platform. For the sake of incomplete information at each step, they use backward induction analysis to drive the Stackelberg equilibrium. Thus, they solve the third problem by leveraging Lyapunov optimization, the second problem by modeling as a bounded knapsack problem and proposing a polynomial approximation algorithm and the first problem by designing an online pricing algorithm based on Zinkevich's online gradient learning approach. Similarly, in [35] authors propose a three-stage Stackelberg game to motivate the users with unknown qualities to participate in a Spatial Crowdsourcing (SC) system. They model the worker selection process as a K-arm Combinatorial MultiArmed Bandit problem and propose a greedy arm-pulling to find the optimal workers. Then, they design the game to determine the optimal strategy group of the payment problem, in which the requester and the platform are the first-tier and second-tier leaders, respectively, while the selected workers are followers and they prove that there exists a unique Stackelberg Equilibrium for the game.

Different from other works, authors in [36] inspired some behavioral economics concepts such as capital deposit, intertemporal choice and addiction to present a long-term motivation. They proposed an addiction incentive mechanism that consists of three steps: the formation, cultivation and maintenance of addiction. This mechanism influences the utility function of participants and motivates them to cooperate in the MCS system for a long-term.

B. RESOURCE MANAGEMENT IN AI APPLICATIONS

The significant growth of AI and Deep Learning pervasiveness specially in smart city with AR technology, caused that rather than continuing to rely on a traditional data center compute model, the industry has embraced the notion of a compute continuum which means putting the right compute resources at optimal processing points in the system spanning from cloud data center to edge systems and endpoint devices. For this reason, AI application partitions placement in computing continuum is gaining increasing interest in research and industrial applications. For example, [10] proposed a hierarchical AI learning framework for the Mobile-Edge-Cloud (MEC) computing paradigm, which develops a novel hybrid method to partition and assign the DNN model layers and the data samples across the three levels of the edge device, edge server and cloud data center. The authors formulate the problem of scheduling the DNN training tasks at both layer partitioning and data partitioning as an integer linear problem (ILP) and they achieve the minimum training time by solving this optimization problem. Conversely, [7] focused on inference time and proposed a lightweight scheduler, called Neurosurgeon to automatically partition DNN computation between mobile devices and data centers. At design time, Neurosurgeon profiles the mobile device and the server to generate performance prediction models of each DNN layer and estimates the power consumption of executing layers on the mobile device. Then at runtime, it selects the best partition point based on the target. If the target is latency/energy, it finds the point that minimizes the total execution-time/energy-consumption. Similarly, Huang et al. [37] assume that there is only one partitioning point and the DNN model is partitioned into two parts for executing DNN inference on mobile web and the edge, respectively. They formulate the partition placement problem as a multi-objective optimization with latency and mobile energy as two objectives while guaranteeing an upper bound constraint for each. They use the weighted sum of latency and mobile energy to solve the problem in linear complexity and find the optimal partition point. Authors in [17] formulated the DNN partitions placement and resource selection problem as a mixed-integer nonlinear program and proposed a random greedy algorithm to solve it.

Some other work on DNN partitions placement considers also the goal of optimizing both training and inference time. For instance, [11] proposed a lightweight DNN with binary neural network (BNN) to execute the DNN on the mobile web with the goal of reducing the size of DNN and accordingly mobile energy consumption and latency. Then, it provides a training method for the lightweight DNN and developed an inference library to run the BNN. Finally, an online scheduler based on Deep Reinforcement Learning

(DRL), is proposed to maximize the resource utilization of the edge devices.

In [12], authors modeled a DNN as a DAG and propose an approach to execute the DNN on a mobile device and cloud, cooperatively, and reduce the mobile cloud computing optimal scheduling problem to the shortest path problem. If the scenario does not have any constraint (mobile battery limitations, quality of service constraints), the optimal scheduling can be obtained by solving the shortest path problem in the graph model. For the scenario with Cloud limited resources, they formulate an ILP for both the training and inference phase by computing the execution time for computation and communication both in the mobile device and cloud VM and try to minimize them by considering a bound for the cloud VM execution time to alleviate the server load. For the scenario with QoS constraint, they minimize the required energy consumption in mobile device while meeting a specified deadline for the total time spent.

Only few works used game theory to model the interaction between mobile user devices and edge or cloud to support the AI application execution. Among them, [38] proposed a DNN partitioning and offloading scheme between mobile devices and edge server. Inspired by Neurosurgeon, it generates performance prediction models of each DNN layer and estimates the power consumption of executing layers on the mobile devices and edge servers. A slot model and dynamic pricing strategy for the servers are used to schedule the offloaded tasks in the MEC side and motivate mobile users towards an efficient task scheduling under budget and delay constraints. At the end, two distributed algorithms with certain and uncertain aggregative information based on aggregative game theory are proposed to minimize the users' device cost. In [39], authors divide the DNN to multiple partitions that can be smaller than a single layer and propose a distributed algorithm to offload partitions based on a matching game approach to reduce the total computation time.

Conversely, [40] models IoT devices competing for computing offloading across fog devices as a weighted potential game. They prove that there exists at least one Nash equilibrium point and they proposed a distributed algorithm to solve the task offloading problem.

All the previous researches investigated the problems of incentive mechanism in MCS and resource management in AI applications, separately, and none of them has addressed the two challenges together. In this work, a MCS system is considered in which the MCS platform expects that the users run an AI sensing task consisting of a DNN and the users have to decide to run the DNN on their local device or remotely on the edge or cloud resources.

Differently from [38] that use a slot model and [39] that use a FIFO execution queue to schedule the tasks in the edge server, in this work M/G/1 queue is used to model the edge and cloud resources. Moreover, on the contrary of [39] that does not consider the cost of neither users nor edge servers and [38] that ignore the cost of edge servers, this paper not only takes into account the user cost but also minimizes the MCS platform cost. In this paper edge servers resources are limited and the MCS platform might to use the extra resources in the cloud to guarantee the application performance constraint incurring extra cost to rent cloud VMs. To efficiently solve this problem, the interaction among the MCS platform and the users has been modeled as an Stackelberg game and a closed form has been found to determine an approximated optimal number of edge and cloud resources by relying on KKT conditions.

III. SYSTEM MODEL

In this section, the MCS model is introduced (Section III-A) and the MCS platform profit and users' cost model is formulated in Section III-B and Section III-C, respectively.

A. MOBILE CROWDSENSING MODEL

This section provides an overview on how the MCS system is modeled to run an AI sensing task among end-user's smart devices and to assign the edge and cloud resources to complete the task if it is needed. A mobile crowdsensing system is considered leveraging edge servers and cloud virtual machines (VMs) as shown in Figure 1: a MCS platform includes edge nodes with limited resources, a pool of unlimited VMs can be run in the cloud side which are connected to the edge platform through a fast network with negligible delay. Finally, a large number of mobile users would like to run an AI sensing task with their smart devices which are connected to the edge platform through a network with a non-negligible latency. As an example, a scenario is considered in which the frames coming from the AI sensing task of the smartphones or smart AR glasses are, possibly, partially processed in the field and then sent to the edge platform/cloud. The goal is to split the AI components execution in a way that the execution cost and energy cost can be optimized by finding a trade off between performance and cost.

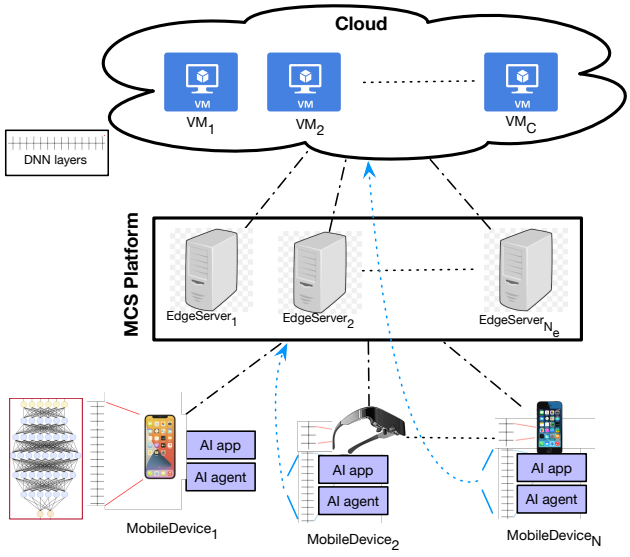


Figure 1: Task's resource assignment model.

The AI sensing task is offered by the MCS platform and a controller agent (see Figure 1) deployed in the end-user's device interacts with the MCS platform and sends some user's information (e.g., battery energy level).

For the sake of simplicity, one sensing task based on a single DNN is considered. A *deployment* contains one or two partitions. A user can decide to run the DNN totally on her/his device or offload a part of computation from her/his device remotely while the MCS platform decides to run deployments locally or deploy and run them on the cloud. It is assumed that the platform knows all users' parameters thanks to the AI sensing task agent and the users have energy and memory constraints to run the task. Moreover, a response time performance constraint is set for the AI task and it must be satisfied.

The set of users is defined as $\mathcal{U}$ and it is assumed that there are N_e homogeneous servers in the edge system unlimited homogeneous VMs (possibly faster than the edge servers) in the cloud while users' devices are heterogeneous.

In the proposed model, the AI sensing task with a DNN can be specified by two alternative deployments indexed by k. An example of AI sensing task with its candidate deployments is shown in Figure 2. The partition point is decided by the platform. This paper, similarly to [7],considers an AlexNet convolutional neural network and the *pool5* layer is selected as the partition point since, according to the profiling data reported in [7], minimizes the end-to-end application latency. Additional details are reported in Section VII-A The first deployment ($k = 1$, see Figure 2a) is a full DNN which can be deployed only on the user device, while the other ($k = 2$, see Figure 2b) have two partitions: the first partition will be run only by the user device while the second one can be run on the edge servers or on the cloud VMs.

The power consumption for running deployment k in user device is denoted by $p_i^{(k)}$ while the power consumption of the second deployment on the edge server is given by p_e. The users receive a time unit reward equal to $U^{(1)}$ when running the task locally (set by the MCS platform which varies in the interval $[U_{min}, U_{max}]$), while receive $U^{(2)} = \gamma U^{(1)}$ for the second deployment $k = 2$, where $\gamma < 1$ because of lighter computational load of the second deployment on the local devices. By considering these rewards for users we can incentivize them to run the task as far as the available memory and energy allow.

$D_i^{(k)}$ is defined as the demanding time (i.e., the time required to serve a single request of the underlying service without resource contention [41]) required to run the deployment $k = 1, 2$ on user device, while the demanding time to serve deployment $k = 2$ is denoted with D_e and D_c on the edge and cloud resources, respectively. Both the edge server and cloud VMs are modelled as an M/G/1 queue [16], shown in Figure 3, while the local end-user's devices are modelled as a delay center. A controller located in the edge side decides to send the load to edge servers or cloud VMs, based on the MCS platform decision.

In this model, the MCS platform would like to maximise its net income given by the revenues to support the sensing task minus the rewards to the users, the energy costs to run the deployments at the edge servers and the cost of remote cloud VMs. Resource planning is performed periodically on a time horizon T. The MCS platform can serve the deployments on the edge by allocating n_e nodes (where $n_e < N_e$, i.e., the number of edge server available) or on cloud with n_c VMs. It is assumed that the MCS platform owns the edge infrastructure and will try to maximize usage of edge resources before leasing any additional resources on a public cloud.

On the other hand, user i decides which deployment to use according to his/her energy consumption for the execution of the deployment and also the reward recived by the MCS for running the deployment. Hence, the mobile users trade-off the energy consumption of their device and the reward received from the platform side. This definition of our system allows us to model the problem as a Stackelberg game in which the MCS platform is the leader and the users are the followers.

In order to define the assignment decisions, namely to characterize which candidate deployment is selected and how the corresponding deployments are assigned to the resources, the following binary variables are introduced:

$$x_i^{(k)} = \begin{cases} 1 & \text{if user } i \text{ selected deployment } k, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

$$y_i^e = \begin{cases} 1 & \text{if user } i \text{ is served by an edge server,} \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

$$y_i^c = \begin{cases} 1 & \text{if user } i \text{ is served by cloud VMs,} \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where $x_i^{(k)}$ is a user decision variable, while y_i^e and y_i^c are the MCS platform decision variables.

There is the following constraint for $x_i^{(k)}$ (at most, only one deployment has to be selected, since the user can decide to stop running the task):

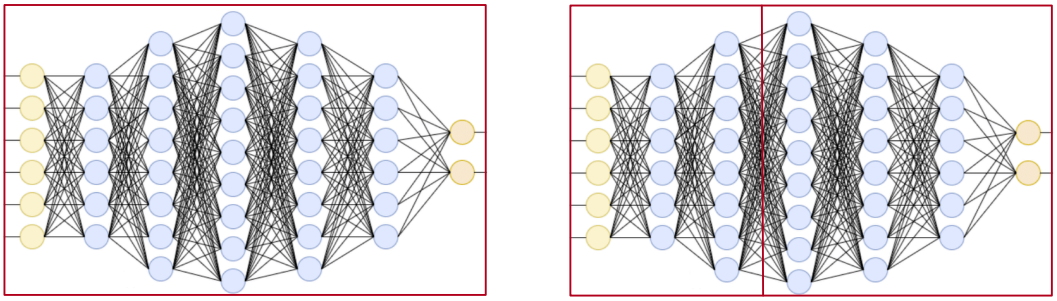
$$\sum_{k=1}^{2} x_i^{(k)} \leq 1, \qquad \forall\, i \in \mathcal{U}, \tag{4}$$

and there is also the following constraint that forces the platform to run the selected deployment of the user either on the edge or on the cloud:

$$y_i^e + y_i^c = x_i^{(2)}, \qquad \forall\, i \in \mathcal{U}. \tag{5}$$

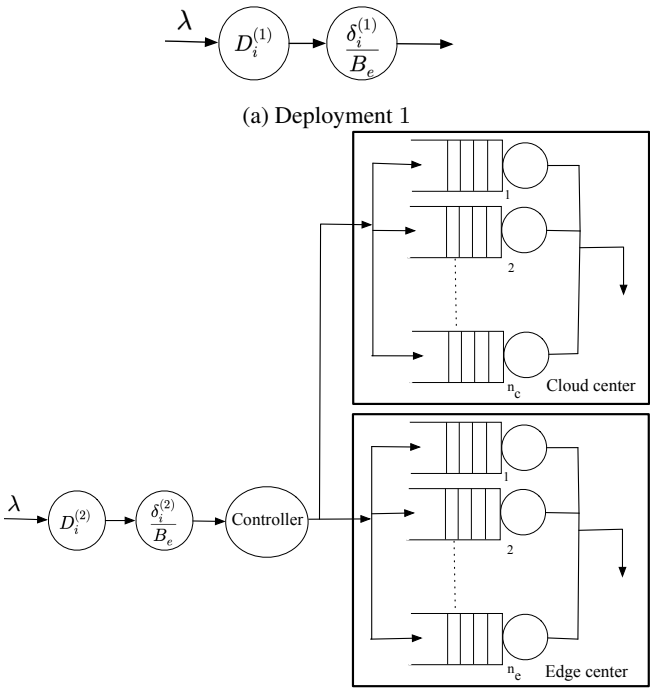
Note that the index of deployments in the previous equation is $k = 2$ because the first deployment will be run totally on the user's device. Each user has a maximum memory capacity denoted by $\bar{M}_i$ which limits the user to run different deployments. Then, the memory constraint can be expressed as follows:

$$\sum_{k=1}^{2} m^{(k)} x_i^{(k)} \leq \bar{M}_i, \tag{6}$$



(a) Candidate deployment 1 (full component) (b) Candidate deployment 2

Figure 2: Example of AI sensing task component with its candidate deployments



(a) Deployment 1

(b) Deployment 2

Figure 3: Queuing model of resources.

where $m^{(k)}$ denotes the memory requirement of deployment k.

The input load of user i is denoted by λ (expressed in terms of requests/s). So the total load of deployment $k = 2$ is as follows:

$$\Lambda = \sum_{i \in \mathcal{U}} \lambda \, x_i^{(2)}. \tag{7}$$

Since all users are going to run the same sensing task, the same input load is considered for all users.

In this work, each edge server and cloud VM is modeled as single server single class queue system (i.e., as an individual M/G/1 queue). Hence, the load of the second deployment in the edge and cloud will be as follows:

$$L_e = \sum_{i \in \mathcal{U}} D_e \, x_i^{(2)} \, \lambda \, y_i^e, \tag{8}$$

$$L_c = \sum_{i \in \mathcal{U}} D_c \, x_i^{(2)} \, \lambda \, y_i^c. \tag{9}$$

In order to avoid resource saturation, if the second partition of deployment $k = 2$ is run at the edge or cloud, the equilibrium conditions for the M/G/1 queue must hold. In particular, this is equivalent to prescribe:

$$\frac{L_e}{n_e} < 1, \tag{10}$$

$$\frac{L_c}{n_c} < 1. \tag{11}$$

This implication can be rewritten as the following alternative conditions:

$$L_e < n_e, \tag{12}$$

$$L_c < n_c. \tag{13}$$

Now, the average response time for user i is computed as follows:

$$R_i = \sum_{k=1}^{2} \left(D_i^{(k)} x_i^{(k)} + \frac{\delta^{(k)} x_i^{(k)}}{B_i} \right) + \frac{D_e x_i^{(2)} y_i^e}{1 - \frac{L_e}{n_e}} + \frac{D_c x_i^{(2)} y_i^c}{1 - \frac{L_c}{n_c}}, \tag{14}$$

where the first expression denotes the average running time of selected deployment on the edge server, the second expression indicates the latency from users to MCS platform, $\delta^{(k)}$ denotes the data transfer size of the first deployment ($k = 1$) or first part of the second deployment ($k = 2$) from user i to the edge platform and B_i indicates the bandwidth of the network. In the above equation, only the transmission delay between the end users and the edge platform is considered, which includes the radio interface. The delay between the edge nodes and the central cloud is ignored because that network segment has high-speed links and is rarely the bottleneck [42]; therefore the resulting delay is negligible. The third and forth expression indicate the average running time of the second deployment on edge and cloud, respectively. It is assumed that $D_e \lambda < 1$ which means that one server in the edge is powerful enough to serve the load of a single user. Moreover, the cloud VMs are more powerful than the edge servers and it holds $D_c < D_e$.

In order to guarantee QoS requirements of the sensing task, a threshold $\bar{R}$ is defined for the average response time of a user:

$$R_i \leq \bar{R}, \qquad \forall\, i \in \mathcal{U}. \tag{15}$$

B. MCS PLATFORM PROFIT MODEL

It is assumed that the platform revenue function to support the sensing task is the strictly concave function defined in (16), where μ is a system parameter. This logarithmic function is widely used in previous works for MCS systems (see, e.g., [25], [26], [28], [43]–[45]) and it is based on the participation of users:

$$\phi(\mathbf{x}) = \mu \log\left(1 + \sum_{i=1}^{N} \log\left(1 + \sum_{k=1}^{2} x_i^{(k)}\right)\right). \tag{16}$$

The rationale behind of this logarithmic function is that, when the number of participants is small, a small increase makes the function increase fast. Then, when the number of participants is larger, the effect slows down. So, this logarithmic function reflects the platform's diminishing return on participating users. The profit of the edge platform is as follows:

$$P_e = \phi(\mathbf{x}) - \sum_{i=1}^{N}\sum_{k=1}^{2} U^{(k)} x_i^{(k)} - T\left(\beta_e p_e n_e + c n_c\right), \tag{17}$$

where β_e is a coefficient to convert the energy consumption of edge server to monetary, c is the cloud VM cost per second (per second billing option is recently available in, e.g., AWS[1] and Azure[2]) and T is time horizon with which the edge provider manages the resource planning. In practice, the impact of the sensing task computation complexity is applied to the cost function. The more computation users do on their device the lower is the cost for the platform because the platform needs less computing resources in the edge and in the cloud.

As stated previously, it is assumed that the cost of running the deployment on the edge is always less than the cloud one, i.e., $\beta_e\, p_e < c$.

C. USER COST MODEL

The energy consumption per request for user i is computed by:

$$T_i \sum_{k=1}^{2} p_i^{(k)} x_i^{(k)}, \tag{18}$$

where T_i denotes the total time user i devotes to run the sensing task. The cost function of user i to run her/his sensing task is as follows:

$$C_i = \lambda\, T_i \beta_i \sum_{k=1}^{2} x_i^{(k)} p_i^{(k)}, \tag{19}$$

where β_i is a coefficient to convert the energy consumption of user's device to monetary cost.

[1]https://aws.amazon.com/ec2/pricing/

[2]https://azure.microsoft.com/en-us/pricing/details/virtual-machines/windows/

IV. PLATFORM AND USER PROBLEM FORMULATION

In the system under study, the MCS problem is modeled as a Stackelberg game where the platform is the leader and the users are the followers. The platform's problem is formulated in Section IV-A and the users' problem in Section IV-B.

A. PLATFORM'S PROBLEM

The MCS platform will solve the following optimization problem:

$$\max_{U^{(1)}, n_e, n_c, y_i^e, y_i^c} P_e = \phi(\mathbf{x}) - \sum_{i=1}^{N}\sum_{k=1}^{2} U^{(k)} x_i^{(k)} - T\left(\beta_e p_e n_e + c n_c\right) \tag{20}$$

subject to:

$$n_e \leq N_e, \tag{21}$$

$$\sum_{k=1}^{2}\left(D_i^{(k)} x_i^{(k)} + \frac{\delta^{(k)} x_i^{(k)}}{B_i}\right) + \frac{D_e x_i^{(2)} y_i^e}{1 - \frac{L_e}{n_e}} + \frac{D_c x_i^{(2)} y_i^c}{1 - \frac{L_c}{n_c}} \leq \bar{R}, \qquad \forall\, i \in \mathcal{U}, \tag{22}$$

$$L_e < n_e, \tag{23}$$

$$L_c < n_c, \tag{24}$$

$$y_i^e + y_i^c = x_i^{(2)}, \qquad \forall\, i \in \mathcal{U}, \tag{25}$$

$$U^{(2)} = \gamma U^{(1)}, \tag{26}$$

$$U_{min} \leq U^{(1)} \leq U_{max}, \tag{27}$$

$$y_i^e, y_i^c \in \{0, 1\}, \qquad \forall\, i \in \mathcal{U}, \tag{28}$$

$$n_e, n_c \in \mathbb{Z}_+. \tag{29}$$

In order to guarantee the problem's feasibility, there are two assumption on the performance constraint:

- If the memory and energy of the device is enough to run the first deployment and $U^{(1)} \geq \lambda T_i \beta_i p_i^{(1)}$, this condition must hold: $D_i^{(1)} + \frac{\delta^{(1)}}{B_i} \leq \bar{R}$;
- If the memory and energy of the device is enough to run the second deployment and $U^{(2)} \geq \lambda T_i \beta_i p_i^{(2)}$, this condition must hold: $D_i^{(2)} + \frac{\delta^{(2)}}{B_i} + \min\{D_e, D_c\} \leq \bar{R}$.

An example of platform net profit function for 100 users (obtained by solving the best reply among the platform and its users) is shown in Figure 4. We set $U^{(1)} = U_{max} = 2$ and then decreased the reward to U_{min} by the step size equal to 0.01. Clearly, the platform problem is not convex. In this analysis it was assumed that all users have enough energy and memory to run both deployments. From $U^{(1)} = 2$ to $U^{(1)} = 0.80$ some linear changes are observed increasingly and decreasingly in platform profit because of changing the selected deployments by users while the platform revenue ($\phi(\mathbf{x})$) is fixed because the number of participants remained fix. When the reward is high, the users select the first deployment because of receiving more reward while by decreasing the reward, gradually, the users select the second deployment because the reward becomes less than the energy cost for running the first deployment. From $U^{(1)} = 0.80$ to $U^{(1)} = 0.30$,

Figure 4: Platform net profit function with 100 users.

the changes are linear increasingly because the users do not change their selected deployment and the first and third terms of platform profit function (20) is fixed while the second term that is linear in $U^{(k)}$ decreases. After $U^{(1)} = 0.30$ some non-linear changes are observed because the users gradually drop and the platform revenue $\phi(\mathbf{x})$ affects on the profit function.

B. USER'S PROBLEM

Each user i will solve the following optimization problem:

$$\max_{x_i} \sum_{k=1}^{2} x_i^{(k)} \left(U^{(k)} - \lambda T_i \beta_i p_i^{(k)} \right) \tag{30}$$

subject to:

$$\sum_{k=1}^{2} x_i^{(k)} \leq 1, \tag{31}$$

$$T_i \sum_{k=1}^{2} p_i^{(k)} x_i^{(k)} \leq \bar{E}_i, \tag{32}$$

$$\sum_{k=1}^{2} m^{(k)} x_i^{(k)} \leq \bar{M}_i, \tag{33}$$

$$x_i^{(k)} \in \{0,1\} \qquad \forall\, k = 1, 2. \tag{34}$$

Constraint (31) guarantees that the user selects at most one deployment, while (32) and (33) avoid running the deployments on the user's local device if it does not have enough energy and memory, respectively. The last constraint defines the decision variable's domain. For the sake of simplicity, parameters and variables are summarized in Table 1 and Table 2, respectively.

V. STACKELBERG GAME FORMULATION

In this section, the Stackelberg game is formulated as a mixed-integer nonlinear optimization problem by embedding the users' problem in the platform problem and by relying on a commercial mixed-integer nonlinear solver. Since finding an optimal solution by a state of the art tool is very slow because of the very large number of variables and constraints

Table 1: Problem Parameters

$\mathcal{U}$	Set of users
N_e	Maximum number of available nodes in the edge.
$D_i^{(k)}$	Demanding time to run deployment k on user i's device.
D_e	Demanding time to run the second deployment on edge server.
D_c	Demanding time to run the second deployment on cloud VM.
$m^{(k)}$	memory requirement of deployment k.
$p_i^{(k)}$	Power consumption of user i's device to run deployment k.
p_e	Power consumption of the edge server to run the second deployment.
λ	Individual user load.
$\delta^{(k)}$	The data transfer size of deployment k from user's device to the edge platform.
B_i	Bandwidth from user i's device to the edge platform.
$\bar{R}$	Upper bound threshold for the average response time of the application.
c	Time unit cost of cloud VMs.
β_i	Coefficient to convert the energy consumption of user i's device to monetary costs.
β_e	Coefficient to convert the energy consumption of the edge server to monetary costs.
$\bar{E}_i$	Maximum energy of user i's device to run the application.
$\bar{M}_i$	Maximum memory of user i's device.
T_i	Total application activation time for user i.
$C_i^{(k)}$	User i's cost to run deployment k.
$s_i^{(k)}$	1 if user i has enough memory and power to run deployment k, 0 otherwise.

Table 2: Decision Variables

$x_i^{(k)}$	1 if user i selected deployment k.
$U^{(k)}$	The reward of users for running deployment k.
R_i	Average response time for user i.
n_e	The number of edge servers to serve the second deployment.
n_c	The number of cloud VMs to serve the second deployment.
$t_i^{(k)}$	Binary variable to show if deployment k is convenient to run on user i device.
$z_i^{(k)}$	Binary variable to show if deployment k is the best deployment to run on user i device.
$x_i^{(k)}$	1 if user i selected deployment k.

for instances of relevance in practice, an efficient heuristic approach to solve the Stackelberg game quickly is proposed in Section VI.

As already mentioned, in our scenario the AI sensing task has a controller agent that provides the knowledge about users' parameters to the MCS platform such as $T_i, \beta_i, \bar{E}_i$ and $p_i^{(k)}$. In this way, the platform can anticipate the user decision in a centralized manner without contacting users. The user side problem in (30)–(34) can be simply solved by checking a few conditions (see Algorithm 1). The necessary condition to run deployment k by a user is that the user has enough energy and memory and the reward received by the platform is more than the cost of running deployment k (lines 3-8). If the problem setting satisfies the user's necessary condition for running all deployments, he/she will select the more profitable one (lines 9-14). Each agent solves the user problem locally, sends the solution to the MCS platform

Algorithm 1 User's MCS algorithm

Input: $i, \alpha_i, p_i^{(k)}, \bar{E}_i, \beta_i, r^{(k)}$
Initialization: $x_i^{(1)}, x_i^{(2)} \leftarrow 0$
if $T_i p_i^{(1)} \leq \bar{E}_i$ **and** $m^{(1)} \leq \bar{M}_i$ **and** $U^{(1)} \geq \lambda T_i \beta_i p_i^{(1)}$ **then**
    $x_i^{(1)} \leftarrow 1$
end if
if $T_i p_i^{(2)} \leq \bar{E}_i$ **and** $m^{(2)} \leq \bar{M}_i$ **and** $U^{(2)} \geq \lambda T_i \beta_i p_i^{(2)}$ **then**
    $x_i^{(2)} \leftarrow 1$
end if
if $x_i^{(1)} = 1$ **and** $x_i^{(2)} = 1$ **then**
    if $U^{(1)} - \lambda T_i \beta_i p_i^{(1)} \geq U^{(2)} - \lambda T_i \beta_i p_i^{(2)}$ **then**
        $x_i^{(2)} = 0$
    else
        $x_i^{(1)} = 0$
    end if
end if
return $x_i^{(1)}, x_i^{(2)}$

that then can solve centrally its problem. The MCS platform problem can then be rewritten as follows:

$$\max_{U^{(1)}, n_e, n_c, y_i^e, y_i^c} P_e = \phi(\mathbf{x}^*(\mathbf{U})) - \sum_{i=1}^{N}\sum_{k=1}^{2} U^{(k)} x_i^{(k)} - T\left(\beta_e p_e n_e + c n_c\right) \tag{35}$$

subject to:

$$\mathbf{x}_i^*(\mathbf{U}) = \arg\max_{\mathbf{x}_i} \left\{ \sum_{k=1}^{2} x_i^{(k)} \left[U^{(k)} - \lambda T_i \beta_i p_i^{(k)} \right] : \sum_{k=1}^{2} x_i^{(k)} \leq 1, \quad T_i \sum_{k=1}^{2} p_i^{(k)} x_i^{(k)} \leq \bar{E}_i, \sum_{k=1}^{2} m^{(k)} x_i^{(k)} \leq \bar{M}_i, x_i^{(k)} \in \{0,1\} \quad \forall\, k = 1,2 \right\}, \quad \forall\, i = 1, \ldots, N, \tag{36}$$

$$n_e \leq N_e, \tag{37}$$

$$\sum_{k=1}^{2} \left(D_i^{(k)} x_i^{(k)} + \frac{\delta^{(k)} x_i^{(k)}}{B_i} \right) + \frac{D_e x_i^{(2)} y_i^e}{1 - \frac{L_e}{n_e}} + \frac{D_c x_i^{(2)} y_i^c}{1 - \frac{L_c}{n_c}} \leq \bar{R}, \qquad \forall\, i \in \mathcal{U}, \tag{38}$$

$$L_e < n_e, \tag{39}$$

$$L_c < n_c, \tag{40}$$

$$y_i^e + y_i^c = x_i^{(2)}, \qquad \forall\, i \in \mathcal{U}, \tag{41}$$

$$U^{(2)} = \gamma U^{(1)}, \tag{42}$$

$$U_{min} \leq U^{(1)} \leq U_{max}, \tag{43}$$

$$y_i^e, y_i^c \in \{0,1\} \qquad \forall\, i \in \mathcal{U}, \tag{44}$$

$$n_e, n_c \in \mathbb{Z}_+. \tag{45}$$

We remark that the optimality constraint (36) can be replaced by a set of linear constraints with additional binary variables. The cost of user i to run deployment k is denoted as parameter $C_i^{(k)}$, so the cost is: $C_i^{(k)} = \lambda T_i \beta_i p_i^{(k)}$. Since the memory and energy constraints are independent of the other variables, these conditions are checked before solving the problem and embedded as binary parameter $s_i^{(k)}$ in the optimization problem. Therefore, binary parameter $s_i^{(k)} = 1$ if user i has both enough energy and memory to run deployment k and $s_i^{(k)} = 0$, otherwise. The linear constraints equivalent to (36) are defined as follows:

$$-(1 - t_i^{(k)})M \leq U^{(k)} - C_i^{(k)} \leq t_i^{(k)} M \qquad \forall\, k \tag{46}$$

$$-(1 - z_i^{(1)})M \leq (U^{(1)} - C_i^{(1)}) - (U^{(2)} - C_i^{(2)}) \leq z_i^{(1)} M \tag{47}$$

$$-(1 - z_i^{(2)})M \leq (U^{(2)} - C_i^{(2)}) - (U^{(1)} - C_i^{(1)}) \leq z_i^{(2)} M \tag{48}$$

$$x_i^{(k)} \leq s_i^{(k)} t_i^{(k)} \qquad \forall\, k \tag{49}$$

$$x_i^{(1)} \geq s_i^{(1)} t_i^{(1)} + s_i^{(2)}(z_i^{(1)} - 1) \qquad \forall\, k \tag{50}$$

$$x_i^{(2)} \geq s_i^{(2)} t_i^{(2)} + s_i^{(1)}(z_i^{(2)} - 1) \qquad \forall k \tag{51}$$

$$x_i^{(1)} + x_i^{(2)} \leq 1 \tag{52}$$

$$x_i^{(k)}, t_i^{(k)}, z_i^{(k)} \in \{0,1\} \qquad \forall\, k \tag{53}$$

where M is a large enough parameter and $t_i^{(k)}, z_i^{(k)}$ are additional binary decision variables. For the reader's convenience, the parameters and variables of users' linear constraints are shown in Table 1 and Table 2.

The Stackelberg game, that is the optimization problem (35)–(45) of the MCS platform side, is a mixed-integer nonlinear program (MINLP) and it can be solved by a global solver. However, not only there is no guarantee to find the optimal solution because of bilinear non-convex term $U^{(k)} x_i^{(k)}$ in the objective function, but also computing the solution is very slow because of the large number of constraints in (38) (as we will show in the experimental result section). Therefore, a heuristic approach is proposed in the next section to solve the problem faster.

VI. HEURISTIC SOLUTION APPROACH

To solve problem instances of practical interest, a heuristic approach based on three main ingredients is developed. First, Section VI-A shows that, under the assumption of a fixed platform reward, a solution to the MCS platform resource allocation problem (i.e., the decisions for n_e and n_c) close to the optimum can be identified by solving a convex optimization problem. Secondly, in Section VI-B it is proved that the optimal reward can be identified by inspection by considering $O(N)$ relevant reward values. Finally, in Section VI-C, a heuristic algorithm is provided to sort the users opting for the second deployment in two groups, one served from the edge and one served from the cloud. Figure 5 indicates the steps of the proposed approach.

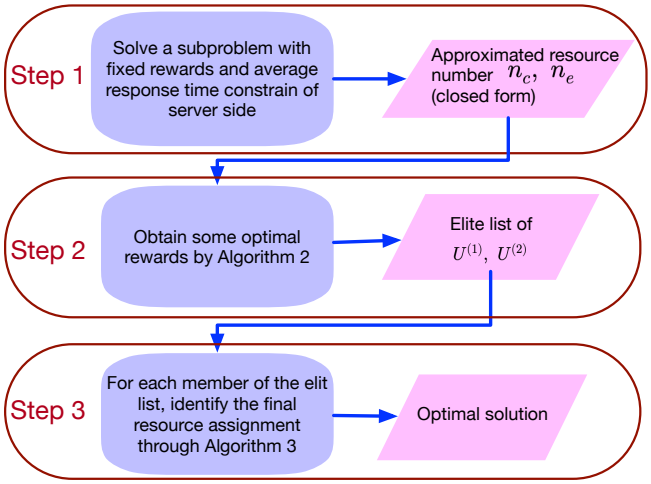


Figure 5: Proposed approach steps.

A. PLATFORM RESOURCE ALLOCATION SUBPROBLEM

This section, shows that the MCS platform resource allocation problem, which consists in finding the optimal values for n_e and n_c, can be approximately reformulated as a convex optimization problem by assuming fixed rewards for the deployments. This solution will then allow to solve the overall Stackelberg game quickly by means of a heuristic approach.

For each possible value of reward, the MCS platform must solve the users' optimization problem by Algorithm 1 and identify each user's desired deployment. Let Λ be the arrival rate of customers opting for deployment $k = 2$.

Next, the MCS platform can compute the amount of edge and cloud resources (n_e, n_c) and how to split Λ into a rate of users that will be served in the edge, λ_e, and in the cloud, λ_c. This is achieved by continuous relaxation as follows.

First, a new response time constraint has been introduced as the response time of platform, $\bar{\bar{R}}$, as follows:

$$\bar{\bar{R}} = \bar{R} - \mathbb{E}\left[D_i^{(2)} x_i^{(2)}\right] - \mathbb{E}\left[\frac{\delta^{(2)} x_i^{(2)}}{B_i}\right], \tag{54}$$

where $\mathbb{E}\left[D_i^{(2)} x_i^{(2)}\right]$ denotes the expected demand time of running the first part of second deployment on the local device of users who selected the second deployment and $\mathbb{E}\left[\frac{\delta^{(2)} x_i^{(2)}}{B_i}\right]$ indicates the expected transmission delay to transfer the output data of the first part of the second deployment to the MCS platform. The rationale behind (54) is that, in order to guarantee the server side response time, it is necessary to provide a margin to account for the user side processing time and data transmission delay. Hence, the response time constraints (22) is redefined on the MCS platform side as follows:

$$\frac{\lambda_e}{\Lambda} \cdot \frac{D_e n_e}{n_e - D_e \lambda_e} + \frac{\lambda_c}{\Lambda} \cdot \frac{D_c n_c}{n_c - D_c \lambda_c} \leq \bar{\bar{R}}. \tag{55}$$

The following result can be demonstrated.

Theorem 1. *The response time constraint in* (55) *is convex.*

Proof. The proof is given in Appendix A. □

In order to identify an estimate for n_e and n_c, a simplified MCS platform problem can be written by neglecting y_i^e and y_i^c, assuming a fixed reward for the deployment, and relaxing the integrality constraint on n_e and n_c, which is now considered as continuous variables:

$$\min_{n_e, n_c, \lambda_e, \lambda_c} \beta_e p_e n_e + c n_c$$

subject to:

$$\begin{aligned}
& n_e \leq N_e, \\
& \frac{\lambda_e}{\Lambda} \cdot \frac{D_e n_e}{n_e - D_e \lambda_e} + \frac{\lambda_c}{\Lambda} \cdot \frac{D_c n_c}{n_c - D_c \lambda_c} \leq \bar{\bar{R}}, \\
& D_e \lambda_e < n_e, \\
& D_c \lambda_c < n_c, \\
& \lambda_e + \lambda_c = \Lambda, \\
& n_e, n_c \geq 0, \\
& \lambda_e, \lambda_c \geq 0.
\end{aligned}$$

Thanks to the linear objective function and the convex constraints, the Karush-Kuhn-Tucker (KKT) optimality conditions can be exploited to obtain the optimal values of decision variable n_e and n_c in the relaxed problem by applying Theorem 2. Integer solutions are then found by rounding.

Theorem 2. *If the total arrival rate satisfies the following condition*

$$\Lambda \leq \frac{N_e(\bar{\bar{R}} - D_e)}{\bar{\bar{R}} D_e},$$

then the MCS platform does not use the cloud VMs ($n_c^* = 0$, $\lambda_c^* = 0$) and the optimal number of edge servers is

$$n_e^* = \frac{\bar{\bar{R}} D_e \Lambda}{\bar{\bar{R}} - D_e}$$

with $\lambda_e^* = \Lambda$.

Otherwise, the edge servers are saturated ($n_e^* = N_e$), the optimal edge rate is

$$\lambda_e^* = \frac{N_e \Lambda(\bar{\bar{R}} - \sqrt{D_c D_e})}{N_e D_e + \bar{\bar{R}} \Lambda D_e - N_e \sqrt{D_c D_e}},$$

the optimal number of cloud VMs is

$$n_c^* = \frac{D_c \Lambda[\bar{\bar{R}} D_e \Lambda - N_e(\bar{\bar{R}} - D_e)]}{N_e(\sqrt{D_e} - \sqrt{D_c})^2 + D_e \Lambda(\bar{\bar{R}} - D_c)},$$

and $\lambda_c^* = \Lambda - \lambda_e^*$.

Proof. The proof is given in Appendix B. □

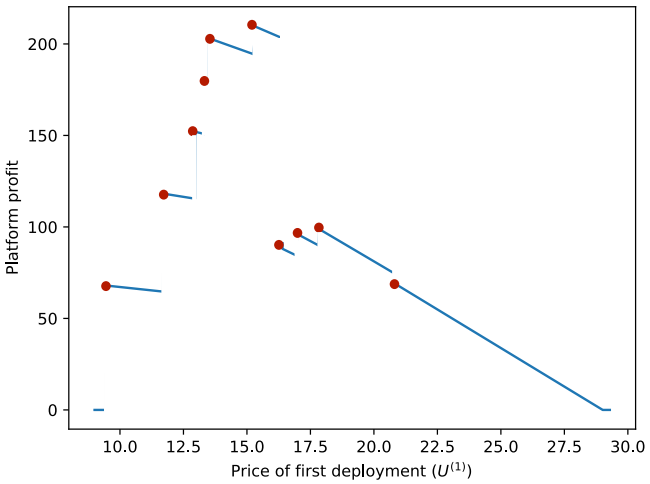


Figure 6: Platform net profit function with 10 users.

B. OPTIMAL REWARD

In the previous section, a fixed reward was assumed for the deployments and a nearly optimal number of resources was found by relying on KKT conditions. In order to solve the Stackelberg game, finding the optimal reward is needed, which we prove being in a finite set of special points. Hence, the optimal value can be found by inspection. Afterwards, Theorem 2 can be applied to find the optimal number of resources for the corresponding optimal reward and then through Algorithm 3, that will be introduced in Section VI-C, the final assign of the users is performed to the resources in the edge/cloud guaranteeing the original response time constraints (22).

To describe the behavior of the platform profit function (mentioned in Figure 4) in details, the profit function related to 10 users scenario is shown in Figure 6. This plot shows that, by varying the reward, the platform profits exhibits points of discontinuity, which are located when a small change in the profit forces users to drop from the system or change their deployment decisions. Accordingly, two types of points of discontinuity are defined:

- **Dropping points:** The values of reward that makes the user's participation benefit equal to zero and it means that the user does not have any motivation to participate in the MCS system. To obtain these points, for each user i, the cost of running the second deployment on the user's device is calculated: $C_i^{(2)} = \lambda T_i \beta_i p_i^{(2)}$. If the reward of running the second deployment ($U^{(2)}$) is equal or less than the cost of user, the user will drop. Therefore, the dropping point of user i is as: $drop_i = U_i^{(2)} = C_i^{(2)}$.
- **Changing deployment points:** These points specify the rewards that make the benefit of running the first and second deployment equal. To obtain these points, for each user i, the cost of running the first and second deployments on the user's device are calculated as: $C_i^{(1)} = \lambda T_i \beta_i p_i^{(1)}$ and $C_i^{(2)} = \lambda T_i \beta_i p_i^{(2)}$, respectively. If the reward of running the deployments are $U_i^{(1)}$ and $U_i^{(2)}$, the user's benefit of running the first and second deployments are: $U_i^{(1)} - C_i^{(1)}$ and $U_i^{(2)} - C_i^{(2)}$. On the other hand, as it was mentioned in Section III-A, the relationship between these rewards is: $U^{(2)} = \gamma U^{(1)}$. Accordingly, the reward that makes the benefits of running two deployment equal for user i can be obtained as follows:

$$U_i^{(1)} = \frac{C_i^{(1)} - C_i^{(2)}}{1 - \gamma} \tag{56}$$

Therefore, changing deployment point for user i is defined as: $change_i = U_i^{(1)}$

We are then able to state the core theorem for our solution as follows:

Theorem 3. *The optimal reward solution lays at one of the points of discontinuity or in a right neighborhood of a point of discontinuity.*

Proof. Assuming the users' decisions $x_i^{(k)}$ are fixed given the platform profit function (20), the only part of the platform profit function that changes with increasing $U^{(1)}$ is $U^{(k)} x_i^{(k)}$, which causes a reduction of the platform profit. So, in each interval in which the users' decisions are fixed, the maximum point is at the beginning of the interval. On the other hand, we know that users' behaviors impose the platform's profit and the behavior of a user will change when they make a decision to participate in the MCS system or not and change the selected deployment from one to another. These behavior changes happen only immediately after the points of discontinuity and the platform profit can increase or decrease as shown in Figure 6. Therefore, in a specific interval between two points of discontinuity where the users' decision are fixed, the optimal solution is either obtained by increasing the reward by ϵ, by setting $U_i^{(1)} = change_i + \epsilon$ or $U_i^{(1)} = \frac{drop_i}{\gamma} + \epsilon$), if the platform profit is also increased.

Otherwise, the optimal solution in that interval is either $U_i^{(1)} = change_i$ or $U_i^{(1)} = \frac{drop_i}{\gamma}$. Accordingly, it can be concluded that the optimal platform profit will happen immediately after or exactly at these points of discontinuity. □

Given Theorem 3, we could just inspect the $O(N)$ points of discontinuity to find the optimal reward (see Algorithm 2). Note, however, that the values for the number of resources n_e and n_c obtained in Section VI-A are not guaranteed to be optimal, because they were obtained by considering in (54) an approximate response time $\bar{\bar{R}}$, which is a function of the average demand time and network transfer delay of users. For this reason, not only the best reward among the points of discontinuity but also an elite set of optimal solutions are considered.

Algorithm 2 receives as input the number of users, the users' parameters, the boundary of rewards and γ. For each user i, the points of discontinuity, which include $drop_i$ and $change_i$ are computed (lines 3-7). Since $drop_i$ is the reward

related to the second deployment, the corresponding reward of first deployment is proportionally equal to $\frac{drop_i}{\gamma}$. Given the reward of the first deployment in discontinuity points and the points immediately after them, if the rewards are in interval $[U_{\min}, U_{\max}]$, the algorithm solves the users' problem according to each reward (U) to obtain the users' participation plan x, finds the approximate n_e, n_c, λ_e, λ_c through Theorem 2 and computes the platform profit and add these results in the solution list (lines 8-17). Finally, the solution list is sorted by P, decreasingly, and the $PopSize$ best elements as a result is returned (lines 18-20).

Algorithm 2 Optimal rewards algorithm

1: **Input:** N, users' parameters, $U_{\min}, U_{\max}, \gamma$
2: **Initialization:** $Solutions \leftarrow []$
3: **for** $i \leftarrow 1$ **to** N **do**
4: $\quad C_i^{(1)} \leftarrow \lambda T_i \beta_i p_i^{(1)}$
5: $\quad C_i^{(2)} \leftarrow \lambda T_i \beta_i p_i^{(2)}$
6: $\quad change_i \leftarrow \frac{C_i^{(1)} - C_i^{(2)}}{1-\gamma}$
7: $\quad drop_i \leftarrow C_i^{(2)}$
8: $\quad$ **for** c **in** $\left\{\frac{drop_i}{\gamma}, \frac{drop_i}{\gamma} + \epsilon, change_i, change_i + \epsilon\right\}$ **do**
9: $\quad\quad$ **if** $U_{min} \leq c \leq U_{max}$ **then**
10: $\quad\quad\quad U^{(1)} \leftarrow c, U^{(2)} \leftarrow c\gamma$
11: $\quad\quad\quad x \leftarrow$ Solve user model given U
12: $\quad\quad\quad n_e, n_c, \lambda_e, \lambda_c \leftarrow$ Compute solution through Theorem 2 given U and x
13: $\quad\quad\quad P \leftarrow$ Compute platform profit given $U, x, n_e, n_c, \lambda_e, \lambda_c$
14: $\quad\quad\quad$ append $(P, U, x, n_e, n_c, \lambda_e, \lambda_c)$ to $Solutions$
15: $\quad\quad$ **end if**
16: $\quad$ **end for**
17: **end for**
18: sort $Solutions$ by P decreasingly
19: **return** first $PopSize$ elements of $Solutions$

C. ASSIGNING THE RESOURCES TO THE USERS

In Section VI-A, closed formulas for the approximate number of edge and cloud resources related to a specific reward have been obtained and Algorithm 2 has been used to maximize the platform profit. The solution obtained by KKT conditions only yields n_e and n_c, then it is needed to assign these resources to the users. This step is necessary for solving the Stackelberg game. The Algorithm 3 is proposed, which solves the assignment problem under the assumption that the load of every user must be served either on the edge or on the cloud (see constraint (5)).

Similarly to Theorem 2, two scenarios are considered. In the first scenario, the edge servers are enough to serve all users' requests, yielding $n_c = 0$. In turn, this scenario has two possible situations: in the first one, the number of edge servers is enough for serving all users and there is no violation of the response time constraint. In the second one, some users' response time constraints are violated. In this latter case, the user with the most severe violations is identified and the number of edge servers that this user needs to avoid violation is computed. In other words, if the user with largest violation is satisfied, we can be sure that all other users are satisfied.

In the second scenario, the number of edge servers is not enough to serve all users' request and cloud VMs are needed, i.e., $n_c > 0$. In this scenario, the users are assigned to the cloud with higher delay induced by running the first partition (on the local device) given the fact that the cloud VMs are faster and the response time violation of such users can be avoided by relying on more performing resources. Then, the required number of edge servers is identified by satisfying the user's response time constraint with the largest value remained in the edge.

Since in both scenarios n_e might have been increased and the maximum number of edge servers overstepped, the users assignment is adjusted by moving the users from edge to the cloud one by one. As a final step, given the load assigned to the cloud, the proper number of cloud resources is identified. These ideas are implemented by Algorithm 3, which receives a list of elite solutions from Algorithm 2, the response time threshold and users' demands as inputs. For each solution in the elite list, given the response time constraint $\bar{R}$, the load of users is appropriately assigned to the resources to obtain an optimal and feasible assignment.

The necessary information is extracted from the initial Algorithm 2 solution (line 4) and a set of users who selected the second deployment is obtained (line 5) then, the two scenarios that was mentioned above is considered. If the first scenario is the case (i.e., the number of edge servers is enough to serve the total load and $n_c = 0$), all users are assigned to the edge and then the response time of all users is evaluated to check the response time constraint violation (lines 6-8).

If no users are in violation, the number of edge servers obtained by the closed form solution of Theorem 2 is the exact result (9-10), otherwise, the violations (V) of all users are sorted increasingly to find the user with maximum violation and compute the minimum number of edge servers ($\bar{n}_e$) that will satisfy his/her response time constraint (lines 11-16).

In the second scenario, the MCS platform needs to use cloud VMs to serve the users' load ($n_c > 0$). In this case, the local execution delay of users (V) is computed (lines 18-20) and the users are sorted by the delay, decreasingly (line 21). Given λ_c and based on the fact that the cloud VMs are faster, first j users are assigned to cloud (lines 21-22) and the others to the edge to avoid the violation of users with higher local delays. Accordingly, the $(j+1)$-th user is the user with maximum local delay in the edge and the minimum number of edge servers ($\bar{n}_e$) is computed to satisfy the $(j+1)$-th user's response time constraint (lines 23-25).

In both scenarios, when the number of edge servers is computed to avoid users' violation, the maximum number of existing servers might have been overstepped in the edge. If this is the case, the user j with maximum V (which can be violation in the first scenario or delay in the second scenario)

is kept in the edge and the assignment of the other users (with lower V) is iteratively changed from the edge to cloud (lines 27-28) until the response time constraint of the user j in the edge is satisfied. In this way, the load of edge and cloud are adjusted (lines 29-30) and after each adjustment the number of edge servers that the user j in the edge needs to avoid violation,is computed (lines 31-32). At the end, given the total load of cloud, the minimum number of cloud VMs, needed to serve the users with maximum V in the cloud, is obtained (lines 34-38). Finally, the platform profit given new optimal number of resources $(\bar{n}_e, \bar{n}_c)$ is computed (line 39) and the solution which maximize the platform profit is returned (lines 40-45).

VII. EXPERIMENTAL RESULTS

In this section, the numerical experiments are presented to evaluate the performance of our approach. The experimental setup and system parameters are introduced in Section VII-A. The scalability analysis and the performance evaluation of the proposed approach compared with the solver and some heuristic baselines are reported in Section VII-B. All experiments were run on a Linux server machine with 40-cores Intel(R) Xeon(R) CPU 2.40GHz and 64 GB memory. BARON 22.3.21 solver has been used and its multithreading option has been enabled to run the solver as fast as possible while the proposed approach was run in a single thread.

All the dataset used in our numerical analyses and our source code are available on Zenodo[3].

A. EXPERIMENT SETUP

In this section, numerical experiments are presented to evaluate the performance of the proposed approach. The system parameters are inspired by the work in [7] where an AlexNet convolutional neural network has been profiled on a real hardware mobile platform and in the cloud. Accordingly, in this paper the *pool5* layer, which minimizes the application end-to-end latency, is considered as the partition point of the second deployment. In [7], the mobile device uses the Jetson TK1 mobile platform which is equipped by mobile SoC, Tegra K1 by NVIDIA [46] and used in the Nexus 9 tablet [47] while the cloud VM is equipped with an NVIDIA Tesla K40 GPU. The parameters of users and cloud (such as demand times, power consumption, data transfer size) are adjusted randomly in a small range $\pm 10\%$ according to the values reported in [7] and the edge side parameters are set proportional to the cloud ones. The parameters (summarized in Table 3) have been set as follows:

- *Edge and cloud demand time:* Cloud demand time is set between 3 and 5 ms and the edge demand is set proportionally equal to $\frac{6}{5}D_c$.
- *Requests demand time:* Users demand time of first deployment is set between 70 and 90 ms while the second deployment is set between 20 and 40 ms.

[3]https://doi.org/10.5281/zenodo.6617705

Algorithm 3 Users' resource assignment

1: **Input:** $EliteSols, \bar{R}, D_i^{(k)}$
2: **Initialization:** $BestProfit \leftarrow 0, y_i^e, y_i^c \leftarrow 0$
3: **for all** Sol **in** $EliteSols$ **do**
4: $U, x, n_e, n_c, \lambda_e, \lambda_c \leftarrow Sol$
5: $N^{(2)} = \{i \in \mathcal{U} : x_i^{(2)} = 1\}$
6: **if** $n_c == 0$ **then** ▷ First scenario
7: $y_i^e \leftarrow 1, \forall i \in N^{(2)}$
8: $V[i] = D_i^{(2)} + \frac{\delta^{(2)}}{B_i} + \frac{D_e}{1-L_e/n_e} - \bar{R}, \forall i \in N^{(2)}$
9: **if all** $V[i] \leq 0$ **then**
10: $\bar{n}_e \leftarrow n_e, \bar{n}_c \leftarrow n_c$
11: **else**
12: sort V increasingly
13: $j \leftarrow$ User with maximum V
14: $LD \leftarrow \bar{R} - D_j^{(2)} - \frac{\delta^{(2)}}{B_j}$
15: $\bar{n}_e \leftarrow \left\lceil L_e \frac{LD}{LD - D_e} \right\rceil$
16: **end if**
17: **end if**
18: **if** $n_c > 0$ **then** ▷ Second scenario
19: $V[i] = D_i^{(2)} + \frac{\delta^{(2)}}{B_i}, \forall i \in N^{(2)}$
20: sort the users by V decreasingly
21: $j \leftarrow \frac{\lambda_e}{\lambda} + 1$
22: $y_i^c \leftarrow 1, \forall i \in N^{(2)} : i < j$
23: $y_i^e \leftarrow 1, \forall i \in N^{(2)} : i \geq j$
24: $LD \leftarrow \bar{R} - D_j^{(2)} - \delta^{(2)}/B_j$
25: $\bar{n}_e \leftarrow \left\lceil L_e \frac{LD}{LD - D_e} \right\rceil$
26: **end if**
27: **while** $\bar{n}_e > N_e$ **do** ▷ Adjusting the load to fix the violation of max edge servers
28: Assign user i with lowest V in edge to cloud $(y_i^e \leftarrow 0, y_i^c \leftarrow 1)$
29: $L_c \leftarrow L_c + D_c\lambda, \lambda_c \leftarrow \lambda_c + \lambda$
30: $L_e \leftarrow L_e - D_e\lambda, \lambda_e \leftarrow \lambda_e - \lambda$
31: $LD \leftarrow \bar{R} - D_j^{(2)} - \delta^{(2)}/B_j$
32: $\bar{n}_e \leftarrow \left\lceil L_e \frac{LD}{LD - D_e} \right\rceil$
33: **end while**
34: **if** $\lambda_c > 0$ **then** ▷ Compute the number of cloud VMs
35: $j \leftarrow$ user with maximum V in cloud
36: $LD \leftarrow \bar{R} - D_j^{(2)} - \delta^{(2)}/B_j$
37: $\bar{n}_c \leftarrow \left\lceil L_c \frac{LD}{LD - D_c} \right\rceil$
38: **end if**
39: $P \leftarrow$ Compute platform profit given $\bar{n}_e, \bar{n}_c$
40: **if** $P > BestProfit$ **then**
41: $BestProfit \leftarrow P$
42: $BestSol \leftarrow (P, U, \bar{n}_e, \bar{n}_c, y^e, y^c)$
43: **end if**
44: **end for**
45: **return** $BestSol$

Table 3: Simulation parameters

System Parameters	
N	In the range (50, 1000)
λ	Uniform distribution in [20, 70] req/s
N_e	$\frac{N}{23}$
D_c	Uniform distribution in [3, 5] ms
D_e	$\frac{6}{5}\ D_c$
$m^{(1)}$	5 MB
$m^{(2)}$	3 MB
β_e	Uniform distribution in [0.2, 0.25] $/kWh
p_e	Uniform distribution in [200, 250] W
$\bar{R}$	Uniform distribution in [0.1, 0.5]
T	3600 s
μ_k	Uniform distribution in [100, 500]
c	Uniform distribution in [1,2] $/h
γ	Uniform distribution in [0.7, 0.8]
ϵ	10^{-8}
$PopSize$	[50, 1000]
$\delta^{(2)}$	Uniform distribution in [0.1, 0.3] MB
$\delta^{(1)}$	$\frac{\delta^{(2)}}{10}$
User Parameters	
$D_i^{(1)}$	Uniform distribution in [(0.07,0.09)]s
$D_i^{(2)}$	Uniform distribution in [0.02,0.04]s
$p_i^{(1)}$	$\frac{9D_i^{(1)}}{0.09}$ W
$p_i^{(2)}$	$\frac{8D_i^{(2)}}{0.04}$ W
β_i	$1.1\beta_e$
T_i	3600 s
$\bar{M}_i$	Uniform distribution in [10, 12] MB
$\bar{E}_i$	Uniform distribution in [5, 7] J
B_i (4G)	Uniform distribution in [5, 10] Mbit/s
B_i (5G)	Uniform distribution in [20, 40] Mbit/s

- *Requests arrival rate* λ: It is set randomly in the range of [20, 70] request per second.
- *Power consumption of edge servers and users:* The power consumption of edge server is determined in range of [200, 250] W, while the power consumption of users' devices is set proportional to their demand time: $p_i^{(1)} = \frac{9D_i^{(1)}}{90\text{ ms}}$ W and $p_i^{(2)} = \frac{8D_i^{(2)}}{40\text{ ms}}$ W.
- *Coefficients* β_e, β_i*:* The industrial electricity cost in Europe [48] is considered, which is approximately in range [0.2, 0.25] $/kWh. β_i is set as 10 percent more than β_e.
- *Cloud VM cost:* The VM cost is determined considering current cloud providers costs for medium range GPU-based VMs and is set in the range [1, 2] $/h.
- *Data size:* The data transfer size of the second deployment is set in the range of [0.1, 0.3] MB and the related amount of first deployment is 10 times less than the second one.
- *Memory:* In all experiments, it is assumed that the memory parameters of users' devices are enough to run both deployments (this settings favour the alternative baselines that are introduced in the following).

To perform the evaluation by considering the existing mobile transmission technologies, a *mixed* scenario has been defined in terms of users' connectivity in which 25% of users are connected through 5G and the others through 4G (according to the predictions this mix is representative for the global market in 2030 [49]).

Moreover, in order to assess the performance of the proposed approach for both small and large systems, the number of users has been varied in the range [50, 1000] increasing the population size with step 50. For a fixed problem instance size, 10 random instances are generated and in the following every method relevant metrics are evaluated as the average across 10 instances.

In order to have a comprehensive comparison, the following baseline methods are defined:

1) *All-Dep1*: In this method, all users will run the first deployment on their own device (recall that parameters are generated in a way memory constraints, see subsection VII-A, are always fulfilled and here the same assumption is initially introduced also for energy) or do not participate. The minimum reward that persuades user i to participate and run the first deployment is

$$U_i^{(1)} = \frac{\lambda T_i \beta_i (p_i^{(1)} - p_i^{(2)})}{1-\gamma}$$

and the best reward of the first deployment (from the platform perspective) is

$$U^{(1)} = \max_{\mathbf{i}\in\mathcal{U}} \left\{ U_i^{(1)} :\ U_{\min} \leq U_i^{(1)} \leq U_{\max} \right\}.$$

2) *All-Dep2*: In this method, it is assumed that all users will run the first partition of the second deployment or do not participate. The MCS platform assign all the participants to the edge/cloud to run the second partition of second deployment. The minimum reward that motivates user i to participate and run the second deployment is

$$U_i^{(2)} = \lambda\, T_i \beta_i p_i^{(2)}$$

and the best reward of the second deployment (from the platform perspective) is:

$$U^{(2)} = \max_{\mathbf{i}\in\mathcal{U}} \left\{ U_i^{(2)} :\ U_{\min} \leq \frac{U_i^{(2)}}{\gamma} \leq U_{\max} \right\}.$$

Assuming the total number of participants is n, to calculate the minimum number of edge servers to serve all participants, it is needed to obtain the maximum delay incurred by running the first partition of second deployment in a local device and transmission delay among all users:

$$Delay = \max_{\mathbf{i}\in\mathcal{U}} \left\{ D_i^{(2)} + \frac{\delta^{(2)}}{B_i} \right\}$$

Depending on assignment of users to edge servers or cloud VMs, the *All-Dep2* method is characterized into two scenarios:

- *All-Dep2-OnlyEdge*: According to the maximum delay among participants, the minimum number of cloud servers to serve all participants will be

$$n_e = \left\lceil n\lambda \frac{\bar{R} - Delay}{\bar{R} - Delay - D_e} \right\rceil .$$

  For the sake of resource limitation in edge, this method might be infeasible if $n_e > N_e$.
- *All-Dep2-OnlyCloud*: Similarly, the minimum number of edge servers to serve all participants will be

$$n_c = \left\lceil n\lambda \frac{\bar{R} - Delay}{\bar{R} - Delay - D_c} \right\rceil .$$

In order to analyse how the proposed approach and baseline methods behave by varying the number of participants, two different scenarios are also introduced in terms of users' device energy: *high energy*, which means all users are able to run both of deployments, and *low energy*, the scenario in which half of users do not have enough energy to run the first deployment but they are still able to run the second deployment. The other half of users can run both of deployments.

To quantitatively evaluate the different methods, we define the gain in terms of platform profit, denoted as *ProfitRatio* as follows:

$$ProfitRatio = \frac{ProposedApproch - OtherMethod}{ProposedApproch},$$

where $OtherMethod$ can denote the profit of BARON or other methods.

B. PERFORMANCE EVALUATION

In this section, numerical experiment is presented to evaluate the proposed approach. First, the results achieved by the proposed approach are illustrated in Section VII-B1 and then, the proposed method is compared with the BARON solver [50] (which has been used to solve the reformulation reported in Section V) and the baseline methods in Sections VII-B2 and VII-B3, respectively.

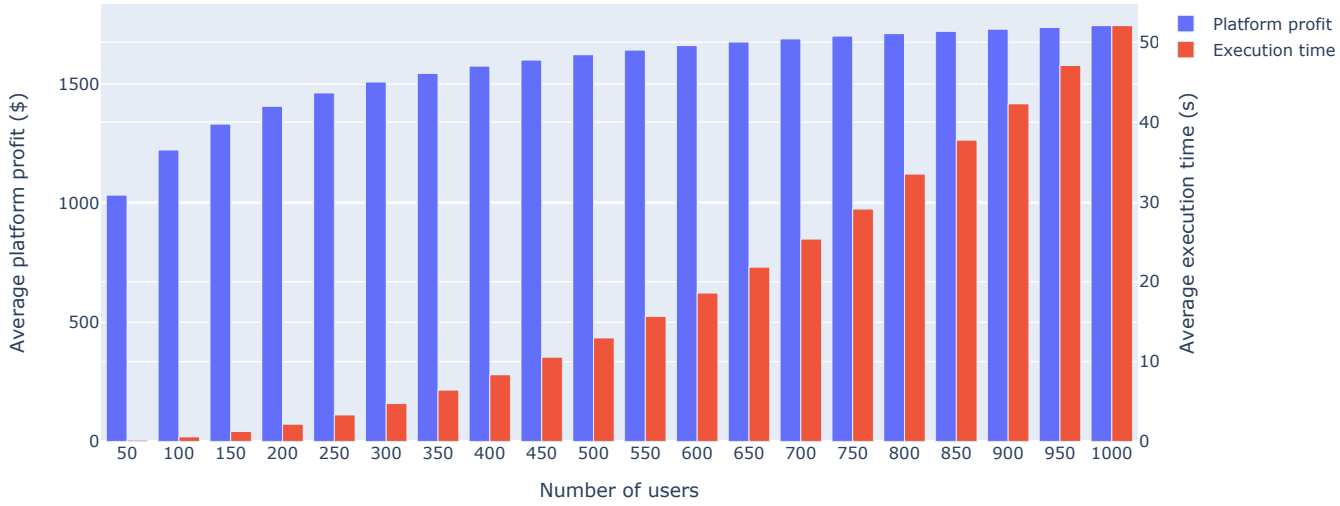


Figure 7: Platform profit and execution time of the proposed approach varying the number of users.

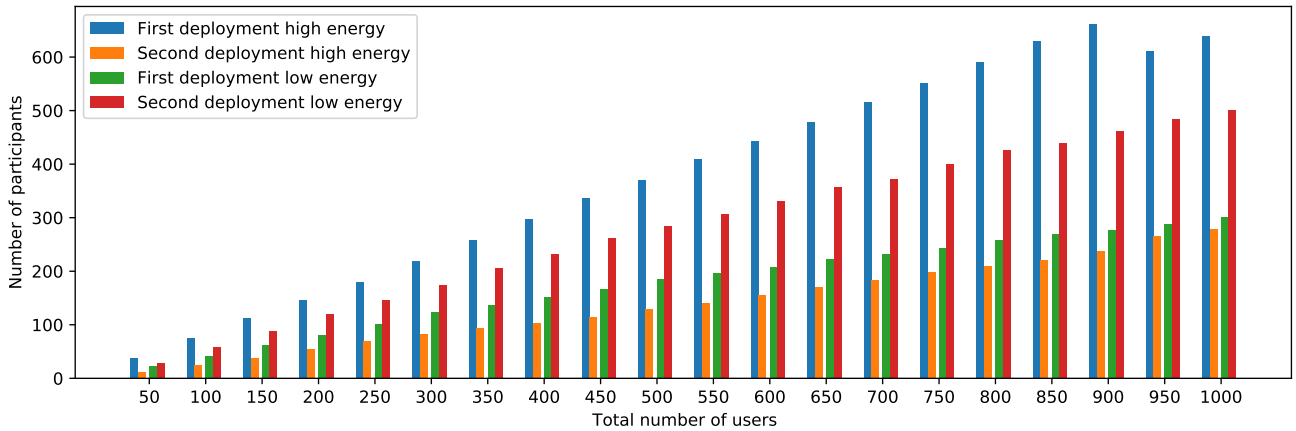


Figure 8: Number of participants and their deployment for different population size.

1) Numerical results of the proposed approach

In this section, the results obtained by the proposed method are illustrated in terms of platform profit, execution time, the number of participants, and the number of resources in use. The result of Figure 7 is related to the *high energy* scenario. The average platform profit achieved by the proposed approach and the average corresponding execution time across 10 instances varying the number of users is shown in Figure 7. The logarithmic relation between the platform revenue and the number of participants (see (16)) is evident in this plot. The maximum execution time that the proposed approach needs to find its best solution in large scale scenarios is less than one minute.

Figure 8 presents the number of participants for each deployment considering the two both energy scenarios by varying the population size for a representative instance (i.e., where demand time, energy, etc. parameters have been fixed while varying the population size). This figure shows that in the *high energy* scenario, the platform best profit will be achieved if most of users run the first deployment vice versa in the *low energy* scenario, if most of users run the second deployment.

Finally, Figure 9 shows the number of edge nodes and cloud VMs used related to the same instances shown in Figure 8. In both energy scenarios, the edge servers are saturated first and then the extra load is assigned to the cloud. In the *low energy* scenario, the platform has to use more cloud VMs because of the inability of users to run the DNN locally on their own devices.

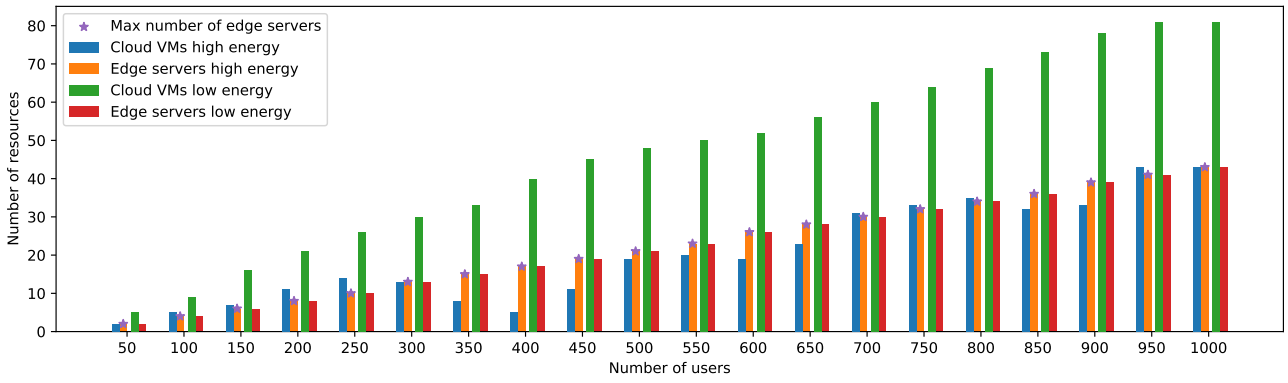


Figure 9: Edge nodes and cloud VMs varying the number of users.

2) Comparison with BARON

To validate the approach, the performance of the heuristic method is compare with the solution of the problem reformulation discussed in Section V, under *high energy* scenario, which can be achieved by BARON solver. Since the problem (35)-(45) has a non-concave objective function and includes non-convex constraints, a global MINLP solver is required to solve this problem. BARON is able to find a nearly optimal global solution through interval analysis and range reduction techniques within a branch-and-bound framework.

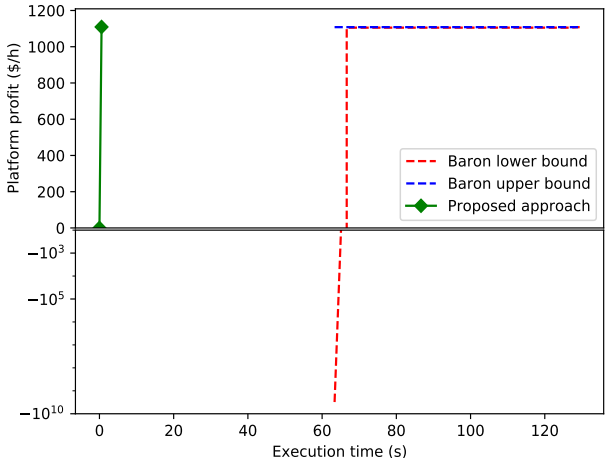
Figure 10 shows, for a representative instance, the lower bound and upper bound identified by BARON vs. the elapsed time (note that the lower bound is the value of the best feasible solution found so far). Since the proposed approach can solve the problem in less than one minute in the worst case with 1000 users, the maximum execution time for BARON has been limited up to one hour. As can be observed in Figure 10a and Figure 10b, BARON found a solution (upper and lower bounds are equal) after about 130s and 2600s for 100 users and 250 users population while the proposed approach can find almost the same solution in 0.5 and 3s, respectively. For 400 users, BARON cannot find any feasible solution for two out of ten instances. Moreover, for feasible instances, BARON does not stop in one hour although the lower bound and upper bound are close (see Figure 10c) while the proposed approach finds the optimal solution in 8s. Finally, BARON cannot find any feasible solution in one hour for 450 users for all 10 instances while the proposed method is able to find the best solution in 10s. Figure 10d shows the lower and upper bounds have not converged. We do not report results for BARON for populations larger than 450 users because we could never obtain a feasible solution within the one hour time limit.

Figure 11 shows the average $ProfitRatio$ across all the feasible instances. As it is shown, BARON loses up to 13% against the proposed approach.

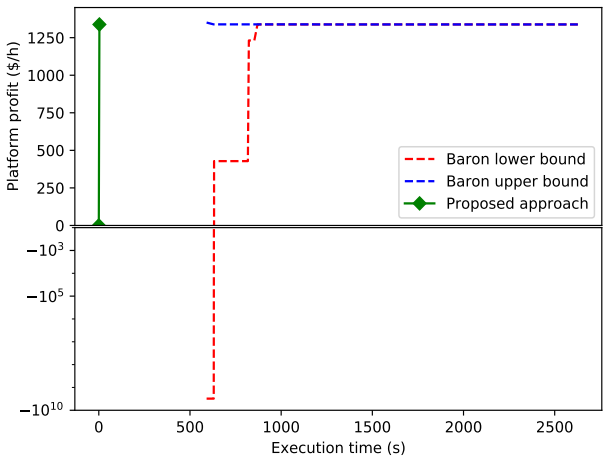
3) Comparison with baseline methods

In this section, the proposed approach is compared with three baseline methods: *All-Dep1, All-Dep2-OnlyEdge* and *All-Dep2-OnlyCloud*. As it is mentioned in Section VII-A, *All-Dep2-OnlyEdge* method might be unfeasible because of the edge resource constraints. Indeed, this method was not able to find any feasible solution for all mentioned scenarios. Accordingly, this method is omitted in the plots of this section.

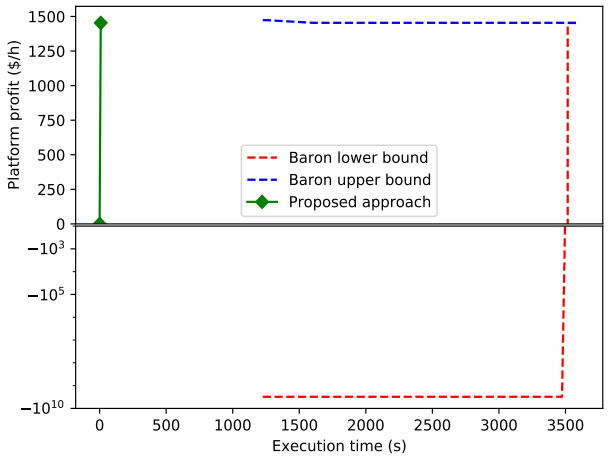
On the other hand, because cloud resources are more expensive than the ege servers nodes, under the *All-Dep2-OnlyCloud* scenario, the platform profit is always negative across all instances. Results are reported in Figure 12.



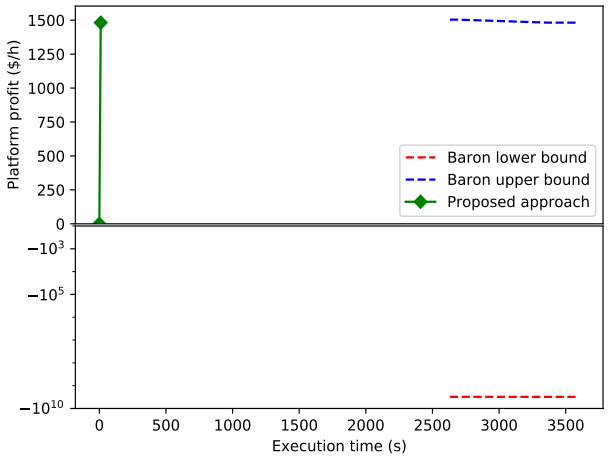
(a) Comparison for 100 users.



(b) Comparison for 250 users.



(c) Comparison for 400 users.



(d) Comparison for 450 users.

Figure 10: Comparison between the solutions obtained by BARON and the proposed approach varying the number of users.

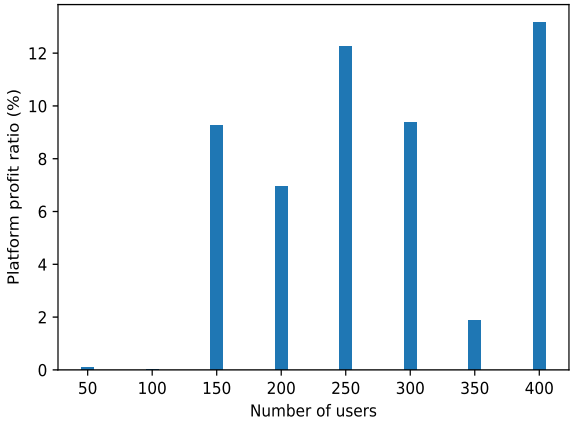


Figure 11: Platform profit ratio obtained when using BARON.

To compare the profit obtained by the proposed approach and *All-Dep1* method, the average $ProfitRatio$ has been calculated across 10 instances of each population size under the two scenarios of *high energy* and *low energy*. Figure 13 shows that when all the users have enough energy to run both deployments, the profit obtained by *All-Dep1* method and our proposed approach are close, the $ProfitRatio$ is about 1% for 50 users while the $ProfitRatio$ increases linearly with the population size up to 8% for 1000 users. The reason for the linear increase is that in *All-Dep1* method in *high energy* scenario, the platform has to pay to all users the reward of first deployment and as it is obvious in platform profit function (see (20)), the relation of platform profit and participants is linear, vice versa in the proposed approach, some users run and receive the reward of the second deployment that is cheaper than the first one while using the shared resources. On contrary, in the *low energy* scenario, only the half of users can participate in *All-Dep1* method and it reduces the platform profit because of the logarithmic profit model especially when the number of users is low, while in the proposed method, all users can participate since the users with energy constraints can still run the second deployment. When the number of users increases, the participation impact gradually decreases by the logarithmic term, then, the profit ratio decreases logarithmically by increasing the

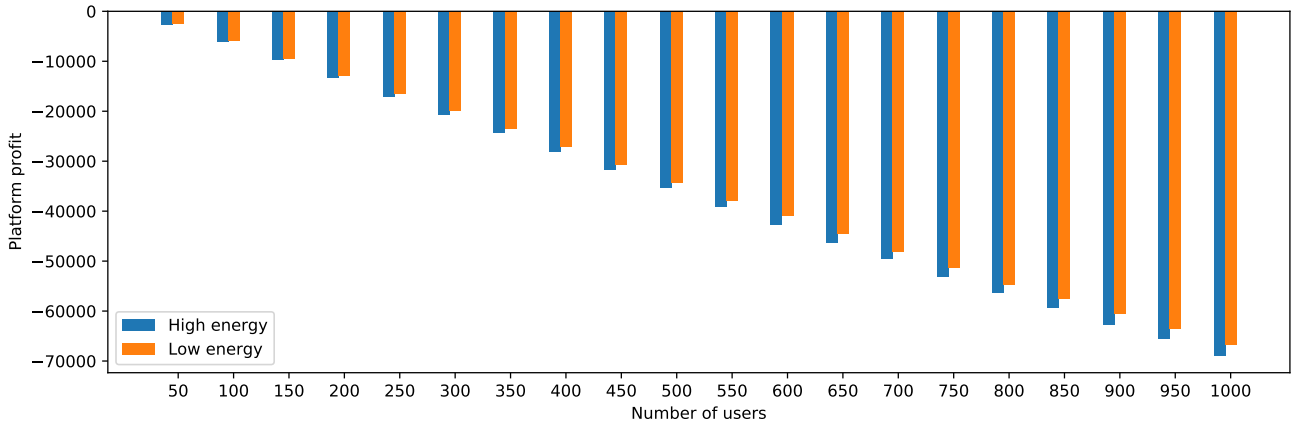


Figure 12: Only cloud scenario profit.

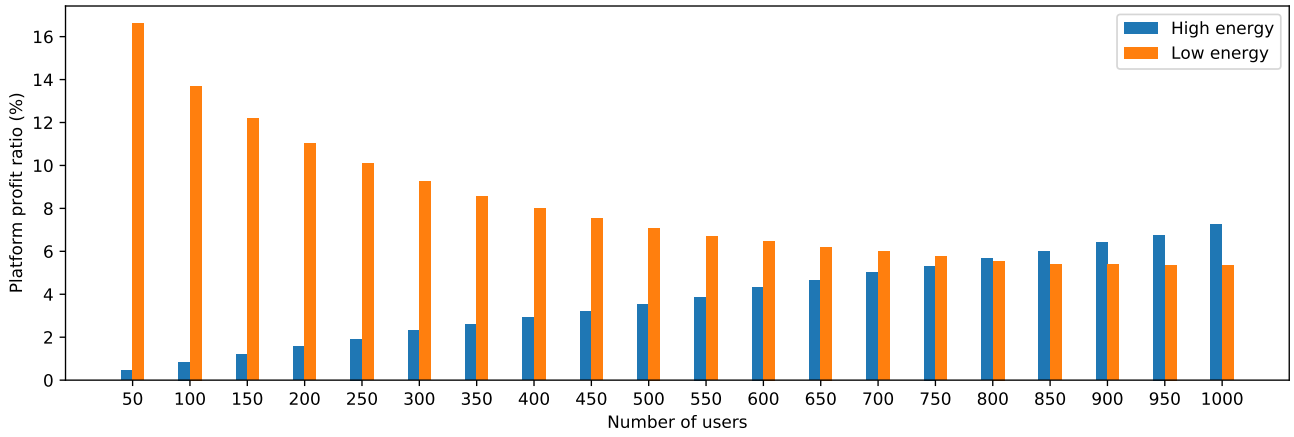


Figure 13: Profit ratio of all deployment 1 scenario.

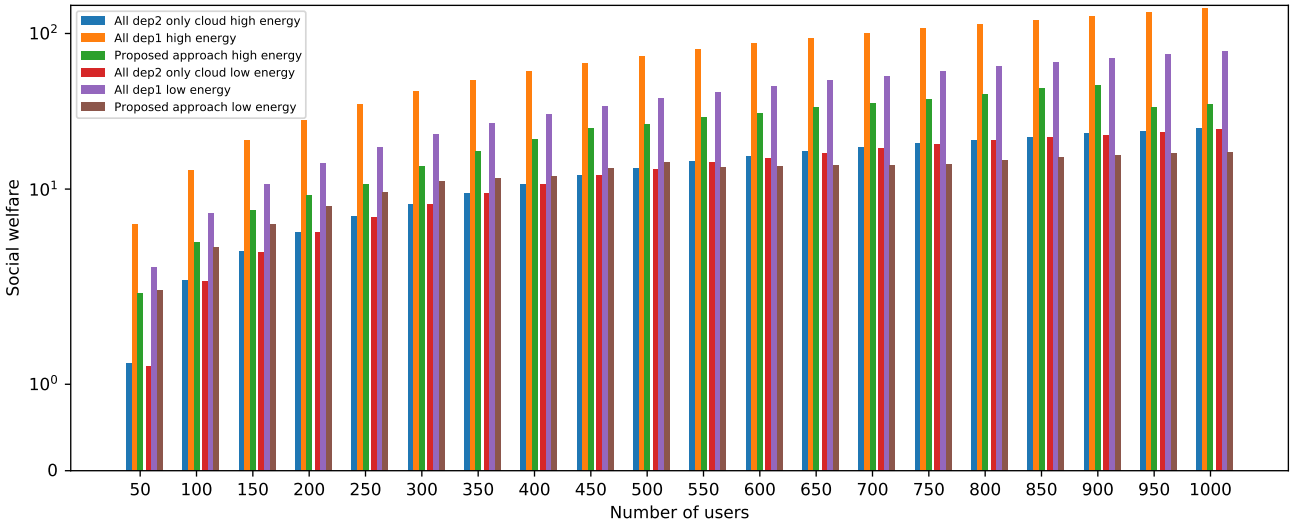


Figure 14: Social welfare comparison.

population size. Overall, under the *low energy* scenario the $ProfitRatio$ ranges in 6-16% while under the *high energy* one the $ProfitRatio$ ranges in 1-8%.

As last analysis, in Figure 14 the average social welfare of users has been computed across 10 instances under both *high* and *low energy* scenarios. *All-Dep1* method always achieves the highest social welfare because of the higher reward of the first deployment vice versa *All-Dep2-OnlyCloud* achieves the

lowest social welfare while the proposed method, privileging the edge platform according to the two level game solution, performs always in the middle.

VIII. CONCLUSIONS

In this paper, an incentive mechanism has been proposed based on Stackelberg games for a mobile crowdsensing system. Different from other literature proposals, an AI sensing task has been introduced based on a DNN application that can be run either totally on users' local device or partially run on the local device and edge or cloud. The problem has been formulated as a MINLP based Stackelberg game and an approach has been developed which is able to find very efficiently an approximated optimal number of edge and cloud resources. Results have demonstrated that the proposed approach outperforms baseline methods under different scenarios and it is order of magnitudes faster than the BARON solver especially in large scale systems.

Future works will extend the proposed method to support multiple partitioning points for the DNN, in a way users have more degrees of freedom to choose the best deployment given their device's energy level and computing capabilities.

IX. APPENDIX

This Appendix provides the proofs of Theorems 1 and 2. The convexity proof of the response time constraint (55) is provided in Appendix A, while the proof of the optimal number of edge servers and cloud VMs is presented in Appendix B.

A. PROOF OF THEOREM 1

Proof. The response time of edge servers for the second deployment is as follows:

$$f(n_e,\lambda_e)=\frac{\lambda_e}{\Lambda}\cdot\frac{D_e n_e}{n_e-D_e\lambda_e}$$

and analyze the convexity of function f on the feasible set $\{(n_e,\lambda_e):\ n_e>D_e\lambda_e\}$.

The first derivative of f respect to n_e is

$$\frac{\partial f(n_e,\lambda_e)}{\partial n_e}=-\frac{(\lambda_e)^2(D_e)^2}{\Lambda\left[n_e-\lambda_e D_e\right]^2}$$

and the second derivative of f respect to n_e is

$$\frac{\partial^2 f(n_e,\lambda_e)}{\partial {n_e}^2}=\frac{2(\lambda_e)^2(D_e)^2}{\Lambda\left[n_e-\lambda_e D_e\right]^3}.$$

The first derivative of f respect to λ_e is

$$\frac{\partial f(n_e,\lambda_e)}{\partial \lambda_e}=\frac{D_e(n_e)^2}{\Lambda\left[n_e-\lambda_e D_e\right]^2}$$

and the second derivative of f respect to λ_e is

$$\frac{\partial^2 f(n_e,\lambda_e)}{\partial {\lambda_e}^2}=\frac{2(D_e)^2(n_e)^2}{\Lambda\left[n_e-\lambda_e D_e\right]^3}.$$

The cross partial derivatives are:

$$\frac{\partial^2 f(n_e,\lambda_e)}{\partial\lambda_e\partial n_e}=\frac{\partial^2 f(n_e,\lambda_e)}{\partial n_e\partial\lambda_e}=-\frac{2\lambda_e(D_e)^2 n_e}{\Lambda\left[n_e-\lambda_e D_e\right]^3}.$$

Therefore, the Hessian matrix of f is

$$\nabla^2 f(n_e,\lambda_e)=\frac{2(D_e)^2}{\Lambda\left[n_e-\lambda_e D_e\right]^3}\begin{bmatrix}(\lambda_e)^2 & -\lambda_e n_e\\ -\lambda_e n_e & (n_e)^2\end{bmatrix}.$$

Since the determinant is equal to zero and the trace is positive, the Hessian matrix is positive semidefinite and f is a convex function on the feasible set.

The same arguments hold for the second term in (55) with respect to the variables n_c and λ_c. Since the response time constraint in (55) is the sum of convex functions, it is convex. □

B. PROOF OF THEOREM 2

Proof. The edge platform problem where no cloud VM is used can be formulated as follows (remind that since it is assumed: $\beta_e p_e < c$, no cloud VMs are used under light load):

$$\begin{aligned}&\min\ \beta_e e_e n_e\\ &\text{subject to: } n_e\le N_e,\\ &\frac{D_e n_e}{n_e-D_e\Lambda}\le\bar{\bar{R}},\\ &n_e-D_e\Lambda>0,\end{aligned}$$

that is equivalent to

$$\begin{aligned}&\min\ \beta_e e_e n_e,\\ &\text{subject to: } n_e\le N_e,\\ &n_e\ge\frac{\bar{\bar{R}}D_e\Lambda}{\bar{\bar{R}}-D_e},\end{aligned}$$

whose optimal solution is

$$n_e^*=\frac{\bar{\bar{R}}D_e\Lambda}{\bar{\bar{R}}-D_e},$$

provided that the feasible region of the latter problem is nonempty, i.e.,

$$\Lambda\le\frac{N_e(\bar{\bar{R}}-D_e)}{\bar{\bar{R}}D_e},$$

that is the total load is small enough.

If the total load does not satisfies the above condition, then edge servers are saturated and also cloud VMs have to be used. Thus, the edge platform problem is:

$$\min_{(\lambda_e,n_c)}\ cn_c$$

subject to:

$$\begin{aligned}&\frac{D_e N_e\lambda_e}{N_e-D_e\lambda_e}+\frac{n_c D_c(\Lambda-\lambda_e)}{n_c-D_c(\Lambda-\lambda_e)}\le\bar{\bar{R}}\Lambda,\\ &0<\lambda_e<\Lambda,\\ &N_e-D_e\lambda_e>0,\\ &n_c-D_c(\Lambda-\lambda_e)>0.\end{aligned}$$

The latter problem is convex and standard constraints qualifications hold (e.g., Slater condition is satisfied), hence it is equivalent to the corresponding KKT system:

$$\mu_1\left[\frac{D_eN_e^2}{(N_e-D_e\lambda_e)^2}-\frac{D_cn_c^2}{(n_c-D_c(\Lambda-\lambda_e))^2}\right]$$
$$-\mu_2+\mu_3+D_e\mu_4-D_c\mu_5=0$$
$$c-\mu_1\frac{D_c^2(\Lambda-\lambda_e)^2}{(n_c-D_c(\Lambda-\lambda_e))^2}-\mu_5=0$$
$$\frac{D_eN_e\lambda_e}{N_e-D_e\lambda_e}+\frac{n_cD_c(\Lambda-\lambda_e)}{n_c-D_c(\Lambda-\lambda_e)}\leq\bar{\bar{R}}\Lambda$$
$$\mu_1\geq 0,\quad \mu_1\left[\frac{D_eN_e\lambda_e}{N_e-D_e\lambda_e}+\frac{n_cD_c(\Lambda-\lambda_e)}{n_c-D_c(\Lambda-\lambda_e)}-\bar{\bar{R}}\Lambda\right]=0$$
$$\lambda_e>0,\quad \mu_2\geq 0,\quad \mu_2\lambda_e=0$$
$$\lambda_e<\Lambda,\quad \mu_3\geq 0,\quad \mu_3(\Lambda-\lambda_e)=0$$
$$N_e-D_e\lambda_e>0,\quad \mu_4\geq 0,\quad \mu_4(N_e-D_e\lambda_e)=0$$
$$n_c-D_c(\Lambda-\lambda_e)>0,\quad \mu_5\geq 0,\quad \mu_5[n_c-D_c(\Lambda-\lambda_e)]=0$$

Notice that constraints imply $\mu_2=\mu_3=\mu_4=\mu_5=0$. Moreover, it follows from second equation that $\mu_1>0$, thus first constraint holds as an equality, i.e.,

$$\frac{D_eN_e\lambda_e}{N_e-D_e\lambda_e}+\frac{n_cD_c(\Lambda-\lambda_e)}{n_c-D_c(\Lambda-\lambda_e)}=\bar{\bar{R}}\Lambda,$$

that is equivalent to

$$n_c=\frac{\left[\bar{\bar{R}}\Lambda-\frac{D_eN_e\lambda_e}{N_e-D_e\lambda_e}\right]D_c(\Lambda-\lambda_e)}{\bar{\bar{R}}\Lambda-\frac{D_eN_e\lambda_e}{N_e-D_e\lambda_e}-D_c(\Lambda-\lambda_e)}. \tag{57}$$

Since $\mu_1>0$, first equation implies that

$$\frac{D_eN_e^2}{(N_e-D_e\lambda_e)^2}=\frac{D_cn_c^2}{(n_c-D_c(\Lambda-\lambda_e))^2},$$

hence

$$\frac{\sqrt{D_e}N_e}{N_e-D_e\lambda_e}=\frac{\sqrt{D_c}n_c}{n_c-D_c(\Lambda-\lambda_e)},$$

that is equivalent to

$$n_c=\frac{\sqrt{D_e}D_cN_e(\Lambda-\lambda_e)}{N_e(\sqrt{D_e}-\sqrt{D_c})+D_e\sqrt{D_c}\lambda_e}. \tag{58}$$

It follows from (57)–(58) that

$$\sqrt{D_c}(N_e-D_e\lambda_e)\left[(N_eD_e+\bar{\bar{R}}\Lambda D_e-N_e\sqrt{D_cD_e})\lambda_e\right.$$
$$\left.-N_e\bar{\bar{R}}\Lambda+N_e\Lambda\sqrt{D_cD_e}\right]=0.$$

Since $N_e-D_e\lambda_e>0$, the optimal edge load is

$$\lambda_e^*=\frac{N_e\Lambda(\bar{\bar{R}}-\sqrt{D_cD_e})}{N_eD_e+\bar{\bar{R}}\Lambda D_e-N_e\sqrt{D_cD_e}}.$$

Finally, the optimal number of cloud VMs from (57) is as follows:

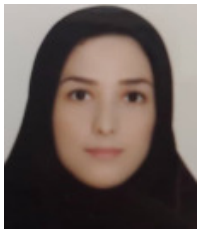
$$n_c^*=\frac{D_c\Lambda[\bar{\bar{R}}D_e\Lambda-N_e(\bar{\bar{R}}-D_e)]}{N_e(\sqrt{D_e}-\sqrt{D_c})^2+D_e\Lambda(\bar{\bar{R}}-D_c)}.$$

□

References

[1] C. Borcea, M. Talasila, and R. Curtmola. Mobile Crowdsensing. Chapman and Hall/CRC, 2016.
[2] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava. Participatory sensing. In The 4th ACM Conference on Embedded Networked Sensor Systems (SenSys), 2006.
[3] V. Le, H. Scholten, and Paul Havinga. Towards opportunistic data dissemination in mobile phone sensor networks. In The Eleventh International Conference on Networks (ICN), 2012.
[4] S.S. Gill, M. Xu, and et al. Ai for next generation computing: Emerging trends and future directions. Internet of Things, 19:100514, 2022.
[5] A. Ghosh, D. Chakraborty, and A. Law. Artificial intelligence in internet of things. CAAI Transactions on Intelligence Technology, 3(4):208–218, 2018.
[6] J. S. Devagiri, S. Paheding, Q. Niyaz, X. Yang, and S. Smith. Augmented reality and artificial intelligence in industry: Trends, tools, and future challenges. Expert Systems with Applications, 207:118002, 2022.
[7] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In ACM ASPLOS '17, 2017.
[8] H. Sedghani, D. Ardagna, M. Matteucci, G.A. Fontana, G. Verticale, F. Amarilli, R. Badia, D. Lezzi, I. Blanquer, A. Martin, and K. Wawruch. Advancing design and runtime management of ai applications with ai-sprint (position paper). In 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), pages 1455–1462, 2021.
[9] E. Li, L. Zeng, Z. Zhou, and X. Chen. Edge ai: On-demand accelerating deep neural network inference via edge computing. IEEE Transactions on Wireless Communications, 19(1):447–457, 2020.
[10] D. Liu, X. Chen, Z. Zhou, and Q. Ling. Hiertrain: Fast hierarchical edge ai learning with hybrid parallelism in mobile-edge-cloud computing. IEEE Open Journal of the Communications Society, 1:634–645, 2020.
[11] Y. Huang, X. Qiao, P. Ren, L. Liu, C. Pu, S. Dustdar, and J. Chen. A lightweight collaborative deep neural network for the mobile web in edge cloud. IEEE Transactions on Mobile Computing, 2020.
[12] A. E. Eshratifar, M. S. Abrishami, and M. Pedram. Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services. IEEE Transactions on Mobile Computing, 20(2):565–576, 2021.
[13] S. Disabato, M. Roveri, and C. Alippi. Distributed deep convolutional neural networks for the internet-of-things. IEEE Transactions on computers, 14(8):1–14, 2015.
[14] S. Teerapittayanon, B. McDanel, and H.T. Kung. Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices. In IEEE 37th ICDCS, 2017.
[15] W. S. Y. Hou, S. Zhou, Z. Niu, Y. Zhang, and L. Geng. Improving Device-Edge Cooperative Inference of Deep Learning via 2-Step Pruning. In IEEE INFOCOM, 2019.
[16] U. Tadakamalla and D. A. Menasce. Autonomic resource management for fog computing. IEEE TCC, pages 1–1, 2021.
[17] D. Ardagna H. Sedghani, F. Filippini. A Random Greedy based Design Time Tool for AI Applications Component Placement and Resource Selection in Computing Continua. In IEEE International Conference on Edge Computing (EDGE), 2021.
[18] X. Zhang, Z. Yang, Z. Zhou, H. Cai, L. Chen, and X. Li. Free market of crowdsourcing: Incentive mechanism design for mobile sensing. IEEE Transactions on Parallel and Distributed Systems, 25(12):3190–3200, 2014.
[19] C. Zhipeng, D. Zhuojun, and L. Wei. Exploiting multi-dimensional task diversity in distributed auctions for mobile crowdsensing. IEEE Transactions on Mobile Computing, 20(8):2576–2591, 2021.
[20] G. Maggie E., A. Ahmad, and B. Ehab F. Maximizing clearance rate of budget-constrained auctions in participatory mobile crowdsensing. IEEE Access, 8:113585–113600, 2020.
[21] X. Mingjun, A. Baoyi, W. Jing, G. Guoju, Z. Sheng, and W. Jie. Cmab-based reverse auction for unknown worker recruitment in mobile crowdsensing. IEEE Transactions on Mobile Computing, pages 1–1, 2021.
[22] D. Xuewen, Y. Zhichao, L. Tom H., Y. Qingsong, S. Yulong, and M. Jianfeng. Optimal mobile crowdsensing incentive under sensing inaccuracy. IEEE Internet of Things Journal, 8(10):8032–8043, 2021.
[23] X. Jia, Z. Yuanhang, D. Yuqing, Y. Dejun, and X. Lijie. Biobjective robust incentive mechanism design for mobile crowdsensing. IEEE Internet of Things Journal, 8(19):14971–14984, 2021.
[24] W. Zhibo, L. Jingxin, H. Jiahui, R. Ju, W. Qian, L. Zhetao, and L. Yanjun. Towards privacy-driven truthful incentives for mobile crowdsensing under

untrusted platform. IEEE Transactions on Mobile Computing, pages 1–1, 2021.
[25] X. Duan, C. Zhao, S. He, P. Cheng, and J. Zhang. Distributed algorithms to compute walrasian equilibrium in mobile crowdsensing. IEEE Transactions on Industrial Electronics, 64(5):195–209, 2017.
[26] Y. Zhan, Y. Xia, and J. Zhang. Incentive mechanism in platform-centric mobile crowdsensing: A one-to-many bargaining approach. Computer Networks, 132:40–52, 2018.
[27] S. He, D. Shin, J. Zhang, J. Chen, and P. Lin. An exchange market approach to mobile crowdsensing: Pricing, task allocation and walrasian equilibrium. IEEE Journal on Selected Areas in Communications, 35(4):921–934, 2017.
[28] Y. Zhan, Y. Xia, and J. Zhang. Quality-aware incentive mechanism based on payoff maximization for mobile crowdsensing. Ad Hoc Networks, 72:44–55, 2018.
[29] H. Sedghani, D. Ardagna, M. Passacantando, M. Z. Lighvan, and H. S. Aghdasi. An incentive mechanism based on a Stackelberg game for mobile crowdsensing systems with budget constraint. Ad Hoc Networks, 123:102626, 2021.
[30] N. Jiangtian, L. Jun, X. Zehui, N. Dusit, and W. Ping. A stackelberg game approach toward socially-aware incentive mechanisms for mobile crowdsensing. IEEE Transactions on Wireless Communications, 18(1):724–738, 2019.
[31] L. Youqi, L. Fan, Y. Song, Z. Pan, Z. Liehuang, and W. Yu. Three-stage stackelberg long-term incentive mechanism and monetization for mobile crowdsensing: An online learning approach. IEEE Transactions on Network Science and Engineering, 8(2):1385–1398, 2021.
[32] L. Xiao, Y. Li, G. Han, H. Dai, and H. V. Poor. A secure mobile crowdsensing game with deep reinforcement learning. IEEE Transactions on Information Forensics and Security, 13(1):35–47, 2018.
[33] Y. Wang, J. Wang, W. Zhang, Y. Zhan, S. Guo, Q. Zheng, and X. Wang. A survey on deploying mobile deep learning applications: A systemic and technical perspective. Digital Communications and Networks, 8(1):1–17, 2022.
[34] Y. Li, F. Li, S. Yang, P. Zhou, L. Zhu, and Y. Wang. Three-stage stackelberg long-term incentive mechanism and monetization for mobile crowdsensing: An online learning approach. IEEE Transactions on Network Science and Engineering, 8(2):1385–1398, 2021.
[35] Y. Xu, M. Xiao, J. Wu, S. Zhang, and G. Gao. Incentive mechanism for spatial crowdsourcing with unknown social-aware workers: A three-stage stackelberg game approach. IEEE Transactions on Mobile Computing, pages 1–1, 2022.
[36] J. Liu, S. Huang, D. Li, S. Wen, and H. Liu. Addictive incentive mechanism in crowdsensing from the perspective of behavioral economics. IEEE Transactions on Parallel and Distributed Systems, 33(5):1109–1127, 2022.
[37] Y. Huang, X. Qiao, S. Dustdar, and Y. Li. AoDNN: An Auto-Offloading Approach to Optimize Deep Inference for Fostering Mobile Web. In IEEE INFOCOM, 2022.
[38] G. Mingjin, S. Rujing, S. Long, Q. Wen, L. Jun, and L. Yonghui. Task partitioning and offloading in dnn-task enabled mobile edge computing networks. IEEE Transactions on Mobile Computing, pages 1–1, 2021.
[39] T. Mohammed, C. Joe-Wong, R. Babbar, and M. Di Francesco. Distributed inference acceleration with adaptive dnn partitioning and offloading. In IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, pages 854–863, 2020.
[40] S. Yu-Jie, W. Hui, and Z. Cheng-Xiang. Balanced computing offloading for selfish iot devices in fog computing. IEEE Access, 10:30890–30898, 2022.
[41] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. Quantitative system performance: computer system analysis using queueing network models. Prentice-Hall, 1984.
[42] Habtegebreil Haile, Karl-Johan Grinnemo, Simone Ferlin, Per Hurtig, and Anna Brunstrom. End-to-end congestion control approaches for high throughput and low delay in 4g/5g cellular networks. Computer Networks, 186:107692, 2021.
[43] T. Liu and Y. Zhu. Social welfare maximization in participatory smartphonesensing. Computer Networks, 73:195–209, 2014.
[44] D. Yang, G. Xue, G. Xue, X. Fang, and J. Tang. Crowdsourcing to smartphones: incentive mechanism design for mobile phone sensing. In Proc. of MobiCom, ACM, page 173–184, 2012.
[45] D. Yang, G. Xue, X. Fang, and J. Tang. Incentive mechanisms for crowdsensing: crowdsourcing with smartphones. IEEE/ACM Transactions on Networking, 24(3):1732–1744, 2016.
[46] NVIDIA Jetson TK1 Development Kit: Bringing GPU-accelerated computing to Embedded Systems. Technical report, 2017.
[47] Nvidia's Tegra K1 at the Heart of Google's Nexus 9, 2016. http://www.pcmag.com/article2/0,2817,2470740,00.asp.
[48] Energy prices in the EU - Statistics & Facts, 2022. https://www.statista.com/topics/4226/energy-prices-in-the-eu/#topicHeader__wrapper.
[49] Bevin Fletcher. High-band 5g to cover 25% of globe by 2030: Mckinsey. Fierce Wireless, 2020. https://www.fiercewireless.com/5g/high-band-5g-to-cover-25-globe-by-2030-mckinsey.
[50] MINLP:BARON solver. https://minlp.com/baron.



HAMTA SEDGHANI received the B.Sc. and M.S degree in Computer Engineering (Software) from Iran University of Since and Technology, Tehran, Iran and university of Tabriz, Tabriz, Iran in 2010 and 2014, respectively. Currently, she is Ph.D. Student in Information Technology Engineering (Computer Networks) and working as a researcher in University of Tabriz. Her current interests include game theory and optimization in mobile crowedsensing.

MINA ZOLFY LIGHVAN received her B.Sc. degree in Computer Engineering (hardware) and M.Sc. degree in Computer Engineering (Computer Architecture) from ECE faculty, university of Tehran, Iran in 1999, 2002 respectively. She received Ph.D. degree in Electronic Engineering (Digital Electronic) from Electrical and Computer Engineering faculty of University of Tabriz, Iran. She is an associate professor in Computer engineering department of ECE faculty in University of Tabriz.

HADI S. AGHDASI received his B.S. degree in Computer Engineering in 2006 from Sadjad University of Technology, Mashhad, Iran and received his M.S. and Ph.D. degrees in computer engineering from Shahid Beheshti University, Tehran, Iran, in 2008 and 2013, respectively. He is an assistant professor of Computer Engineering department at University of Tabriz, Tabriz, Iran. His current researches focus on Humanoid Robots and Intelligent Methods in Surveillance Systems and Cognitive Technology. Also, he works on Wireless Traditional and Visual Sensor Networks (Routing, Clustering, Coverage, and Visual Information Transmission). He is a director of the both Humanoid Robots and Cognitive Technology (HRCT) and Wireless Ad hoc and Sensor networks (WASL) research laboratories in University of Tabriz.

MAURO PASSACANTANDO received the M.S. and Ph.D. degrees in Mathematics from University of Pisa, Italy, in 2000 and 2005, respectively. He is currently Associate Professor of Operations Research (qualified for Full Professorship) at the Department of Computer Science at University of Pisa. His research is mainly devoted to variational inequalities and equilibrium problems. In the last years, his work focused on game theoretic models applied to service provisioning problems in cloud and multicloud systems and infrastructure and spectrum sharing in mobile networks.

GIACOMO VERTICALE received the Ph.D. degree in telecommunications engineering from the Politecnico di Milano, Italy, in 2003. He is currently an Associate Professor at the Politecnico di Milano. He was involved in several research projects on fixed and wireless broadband access technologies and promoting the smart grid. His current interests focus on the security issues of the smart grid, on network function virtualization, and on edge computing in 5G.

DANILO ARDAGNA is Associate Professor at the Dipartimento di Elettronica Informazione and Bioingegneria at Politecnico di Milano. He received a Ph.D. degree in computer engineering in 2004 from Politecnico di Milano from which he also graduated in December 2000. His work focuses on the design, prototype and evaluation of optimization algorithms for resource management of cloud computing and big data systems.