IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 12, 2019, accepted July 17, 2019, date of publication July 25, 2019, date of current version August 14, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2931336

Joint Cotask-Aware Offloading and Scheduling in Mobile Edge Computing Systems

YI-HAN CHIANG[1], (Member, IEEE), TIANYU ZHANG[1,2], AND YUSHENG JI[1,2], (Senior Member, IEEE)

[1]Information Systems Architecture Research Division, National Institute of Informatics, Tokyo 101-8430, Japan
[2]Department of Informatics, SOKENDAI (The Graduate University for Advanced Studies), Tokyo 101-8430, Japan

Corresponding author: Yi-Han Chiang (yhchiang@nii.ac.jp)

This work was supported in part by the Japan Society for the Promotion of Science (JSPS) KAKENHI under Grant 18H06471, Grant 18KK0279, and Grant 19K21539.

ABSTRACT Mobile edge computing (MEC) systems provide mobile devices (MDs) with low-latency cloud services by deploying edge servers (ESs) in the vicinity. In fact, various mobile applications may generate cotasks, each of which is completed only if all its constituent tasks are finished. Existing works have been devoted to the design of MEC offloading and scheduling, but none of them exploits the cotask feature to better utilize the networked computing resources. In this paper, we investigate the problem of joint cotask-aware offloading and scheduling in MEC systems (Cool-Edge), and we formulate it as a mixed integer non-linear program (MINLP), the objective of which is to minimize average cotask completion time (ACCT). To cope with the Cool-Edge problem, we propose two low-complexity algorithms to offload cotasks based on an LP rounding technique and schedule them according to an earliest-cotask-arrival-first rule, respectively, and we further prove the approximation factor jointly achieved by the two algorithms. Finally, we conduct testbed experiments and simulations to demonstrate the effectiveness of our proposed solution, and we also show how ACCT varies with the network environment.

INDEX TERMS Cotask, mixed integer non-linear program, mobile edge computing system, task offloading, task scheduling.

I. INTRODUCTION

Recent advances show that various emerging delay-sensitive applications have proliferated and become parts of mobile devices (MDs) in recent years. However, resource-constrained MDs (due to their limited battery lives and computing resources) may have difficulties in performing computation-intensive jobs while meeting stringent delay requirements. Therefore, mobile edge computing (MEC) systems [1]–[5] came into the world to resolve this problem by deploying edge servers (ESs) on the network edge, thereby providing low-latency cloud services to MDs.

In MEC systems, offloading and scheduling are the essential design factors on MDs and ESs, respectively (see FIGURE 1 for their relationship). MEC offloading is for MDs to determine whether to process their jobs remotely. Since offloading jobs involves sending to and receiving from ESs, such a decision should be adapted to wireless channel characteristics (e.g., [6]–[8]). On the other hand, MEC scheduling is for ESs to determine the order in which the offloaded jobs are processed. Generally, ESs are not as powerful as traditional cloud servers in data centers, and hence they should produce efficient schedules according to their computing resources (e.g., [9]–[11]). Clearly, these two factors are coupled: MEC offloading determines the amount of jobs to be offloaded, while MEC scheduling has influences on how efficiently the offloaded jobs can be processed. Therefore, MEC offloading and scheduling should be jointly designed, instead of being tackled separately.

To fully grasp the potentials of MEC systems, leveraging parallel processing that admits jobs to be processed on independent machines is an effective way to exploit the networked computing resources. Kosta *et al.* [12] propose a framework to facilitate on-demand resource allocation and parallelism for managing virtual machines in the cloud. Jia *et al.* [13] design an online task offloading algorithm for concurrent tasks, where the edge and cloud computation can be performed in parallel. Yang *et al.* [14] investigate the problem of joint computation partitioning and resource allocation, where each user's application is divisible into modules that

The associate editor coordinating the review of this manuscript and approving it for publication was Peng-Yong Kong.

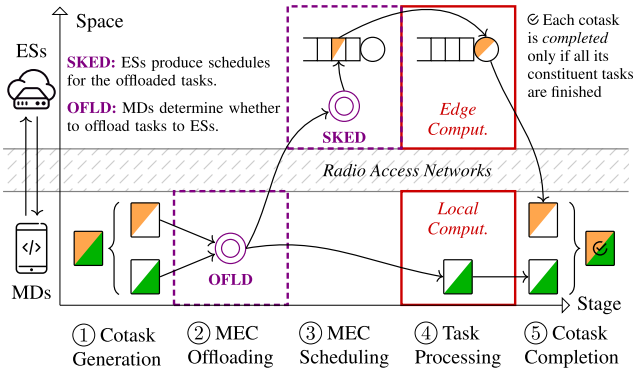


FIGURE 1. The relationship between MEC offloading and scheduling, and the generation and completion of cotasks.

can be processed concurrently. In fact, mobile applications may generate data-parallel jobs (e.g. video navigation [15]) as **cotasks**:

- *each cotask is completed only if all its constituent tasks are finished and returned to the hosting MD*, and
- *the cotask completion time is determined by the latest completion time among all of the constituent tasks*.

Despite the above works that guide us to leverage parallel processing to accelerate computation for MDs, none of them can tell us how to properly allocate computing resources to cotasks in MEC systems.

In this paper, we investigate the problem of cotask-aware offloading and scheduling in MEC systems (Cool-Edge). The Cool-Edge problem can be characterized as a mixed integer non-linear program (MINLP), the objective of which is to minimize average cotask completion time (ACCT). Due to the NP-hardness of the Cool-Edge problem, we are motivated to design approximation algorithms with a provable performance guarantee. For MEC offloading, we consider an LP to minimize the makespan for each cotask, and then apply an LP rounding (LPR) technique that explores a perfect matching from a constructed bipartite graph. For MEC scheduling, we propose to schedule cotasks according to the earliest-cotask-arrival-first (ECAF) rule in the way that the earliest cotask arriving at the network edge will be processed first by all ESs. We conduct testbed experiments to assert the practicability of our proposed solution. In larger network scales, we run simulations to demonstrate the achieved ACCT as well as the impacts of the constitution of cotasks and the distribution of computing resources. The contribution of this paper can be summarized as follows:

- We investigate how the cotask feature matters in the design of MEC offloading and scheduling.
- We formulate the Cool-Edge problem as an MINLP so as to minimize ACCT, and show its NP-hardness.
- We propose two approximation algorithms to offload cotasks based on the LPR technique and employ the ECAF rule for scheduling, respectively.
- The proposed solution achieves an approximation factor of $4\kappa\,(1+\epsilon)+2\rho$ for any $\epsilon > 0$, where κ and ρ are the ratios of machine computability and radio access delays, respectively.
- Both testbed experiments and simulations are provided to show the effectiveness of our proposed solution.

The rest of this paper is organized as follows. In Sec. II, we compare cotasks with coflows and introduce related works on task/job offloading and cotask/coflow scheduling. In Sec. III, we consider an MEC system where MDs generate cotasks. In Sec. IV, we formulate the Cool-Edge problem as an MINLP and show its intractability. In Sec. V, we design two approximation algorithms for cotask-aware MEC offloading and scheduling, respectively, and prove the achieved approximation factor. In Sec. VI, we show our testbed experiments and simulations. Finally, this paper is concluded in Sec. VII.

II. RELATED WORK

Cotasks in MEC systems share similar concepts with coflows (of which each job is composed of multiple constituent flows) in data center networks, where coflow scheduling has lately attracted growing interests due to its practical importance echoed by [16]–[18]. In fact, there are two key differences between coflows and cotasks. First, coflows are mostly synchronous (i.e., they are released for processing at the same time) whereas cotasks in MEC systems are typically asynchronous (i.e., MD-ES pairs are with distinct release time and uplink delays). Second, each coflow is finished only after all its constituent flows are completed. However, the finish of a cotask implies that all the constituent tasks have been processed and also returned to the hosting MD. As a consequence, cotasks in MEC systems will encounter non-zero downlink delays due to returning computational results to the hosting MD, which does not take place on coflows.

In literature, there are a few works discussing the design of offloading and scheduling, part of which are with the cotask/coflow feature. Wang *et al.* [9] study the problem of assignment and scheduling for offloaded tasks in mobile edge clouds to optimize cost efficiency. Tan *et al.* [10] investigate the problem of job dispatching and scheduling in edge-cloud systems to minimize job response time. Mahmoodi *et al.* [15] jointly consider cloud offloading and local scheduling for multi-component applications. Chang *et al.* [16] aim to design scheduling algorithms for MapReduce-like systems to optimize coflow completion time. Zhao *et al.* [19] focus on cotask scheduling design to minimize cotask completion time for cloud computing. Although the former three works provide design principles for joint offloading and scheduling while the latter ones focus on cotask/coflow scheduling, it remains unknown how to jointly design cotask-aware offloading and scheduling for MEC systems.

III. SYSTEM MODEL

We consider an MEC system consisting of a set $\mathcal{K}$ of ESs (interconnected by a controller[1]) and a set of MDs generating cotasks, where we denote by $\mathcal{J}$ the generated cotasks and

[1]The controller is typically deployed in the vicinity of ESs or collocated with an ES, so that it can promptly respond to both MDs and ESs.

TABLE 1. Notation summary.

Notation	Definition
$\mathcal{J}, \Xi_j$	a set of cotasks, the hosting MD of cotask j
$\mathcal{K}, \mathcal{K}_j$	a set of ESs, the set of ESs plus MD Ξ_j
$\mathcal{I}_j$	the set of constituent tasks of cotask j
T_j^{rel}	the release time of cotask j
T_{ijk}^{ul}	the uplink delay of task i from MD Ξ_j to PU k
T_{ijk}^{dl}	the downlink delay of task i from PU k to MD Ξ_j
T_{ijk}^{proc}	the processing time of task i of cotask j on PU k
z_{ijk}	the offloading decision of task i of cotask j to PU k
σ_k	the scheduling decision of offloaded tasks on PU k
τ_{jk}	the completion time of tasks of cotask j via PU k
t_j	the completion time of cotask j
κ	a ratio of machine capability
ϵ	an error-tolerant parameter
ρ	a ratio of radio access delays

Ξ_j represents the hosting MD of cotask j. The controller is clairvoyant (i.e., the processing time of each task can be revealed to the controller), [2] and it guides MDs to make offloading decisions and also persistently updates ESs on the latest schedule. Since both MDs and ESs are capable of processing tasks, we call them processing units (PUs) for generality. We assume that each PU can process at most one task at a time, and it will not migrate its received tasks to any other PUs. For brevity, we let $\mathcal{K}_j$ be the set of PUs to which the cotask j is offloadable, and since we do not allow MDs to offload cotasks in between, it is clear that $\mathcal{K}_j \cap \mathcal{K}_{j'} = \mathcal{K}$, $\forall j' \in \mathcal{J} \setminus \{j\}$. The notations used throughout this paper are summarized in TABLE 1.

Each cotask j has a set $\mathcal{I}_j$ of constituent tasks, each of which can be processed either on MD Ξ_j (local computation) or on an ES (edge computation). In addition, each cotask j is associated with several time information when it is generated. First, the cotask is present to MD Ξ_j at its release time T_j^{rel}. Second, task i of the cotask to PU $k \in \mathcal{K}_j$ will take $T_{ijk}^{\mathrm{ul}} = L_{ij}^{\mathrm{ul}}/R_{jk}^{\mathrm{ul}}$ and $T_{ijk}^{\mathrm{dl}} = L_{ij}^{\mathrm{dl}}/R_{jk}^{\mathrm{dl}}$ for the uplink and downlink transmissions, [3] where $L_{ij}^{(\cdot)}$ and $R_{jk}^{(\cdot)}$ indicate the data size of task i of cotask j and the data rate [4] between MD Ξ_j and PU k, respectively, for $(\cdot) \in \{\mathrm{ul}, \mathrm{dl}\}$. Third, each constituent task i can be processed on any PU k, where the processing time is denoted by T_{ijk}^{proc}.

To understand how cotask awareness matters in the design of MEC offloading and scheduling, we consider two motivating examples as follows. For ease of elucidation, we define the task completion time (TCT) of cotask j via PU k as the time by which the set of tasks that are offloaded to PU k have been finished (i.e., being processed and returned to MD Ξ_j), while the CCT of cotask j is the time by which all of the constituent tasks are finished. In addition, we use the symbols in FIGURE 2a throughout the examples.

[2] If this information is unavailable in the very beginning, it is necessary to first process the task (on all PUs) and then record the measured processing time. The recorded information can then be used for the following tasks that are generated by the same application under similar configurations.

[3] The uplink and downlink delays are the measures that reflect the effect of the sharing of network bandwidth.

[4] When tasks are processed locally, neither uplink nor downlink transmissions will take place. Equivalently, $R_{jk}^{\mathrm{ul}} = \infty$ and $R_{jk}^{\mathrm{dl}} = \infty$ when $k = \Xi_j$.

A. COTASK-AWARE MEC OFFLOADING

(FIGURE 2) We consider one ES and three cotasks, where the release time, the time elapsed for processing locally or remotely, and the uplink and downlink delays are all shown in FIGURE 2b. In this example, it reveals that processing all tasks locally (see FIGURE 2c) loses the chances to speed up processing, and offloading all tasks to ES k_1 (see FIGURE 2d) is also unwise due to the potentially large waiting time on the ES. Once we offload each individual task to a PU that provides the shortest TCT (see FIGURE 2e), the computing resource of ES k_1 could be underutilized. If the design of MEC offloading is aware of the cotask feature (see FIGURE 2f), then a balanced local and edge computation and an elevated ACCT can be anticipated.

B. COTASK-AWARE MEC SCHEDULING

(FIGURE 3) Besides MEC offloading, cotask awareness also matters in MEC scheduling. Here, we consider two ESs and two cotasks (each of which consists of four constituent tasks) as depicted in FIGURE 3a. In FIGURE 3b, we see that the first tasks processed by k_2 and k_3 can be finished and returned to j_4 quickly, but long waiting time takes place on j_4 due to the slow local computation. On the other hand, the offloaded tasks from j_5 are awaiting for processing on both k_2 and k_3, and hence they are finished and returned late. If the cotask feature is considered in MEC scheduling (see FIGURE 3c), unnecessary waiting time on all PUs could be reduced, thereby improving the ACCT.

IV. PROBLEM FORMULATION

In this section, we consider an optimization framework to characterize how cotask awareness affects MEC offloading and scheduling. For this, we describe several decision variables and constraints, followed by the formulated Cool-Edge problem. In addition, we prove the NP-hardness of the Cool-Edge problem, which motivates us to design approximation algorithms with provable performance guarantees.

A. DECISION VARIABLES

To fully describe the behaviors of cotask-aware MEC offloading and scheduling, we define two decision variables:

$$\text{(cotask offloading) } \boldsymbol{z} = \left\{z_{ijk},\ \ \forall i \in \mathcal{I}_j, k \in \mathcal{K}_j, j \in \mathcal{J}\right\},$$
$$\text{(cotask scheduling) } \boldsymbol{\sigma} = \left\{\sigma_k,\ \ \forall k \in \mathcal{K}\right\},$$

where z_{ijk} is a binary variable representing if task i of cotask j is processed by ES k, and σ_k is a one-to-one function that maps $\mathcal{L}_k$ to positive integers as

$$\sigma_k : \mathcal{L}_k \to \{1, 2, \ldots, |\mathcal{L}_k|\}, \quad \forall k \in \mathcal{K}, \tag{1}$$

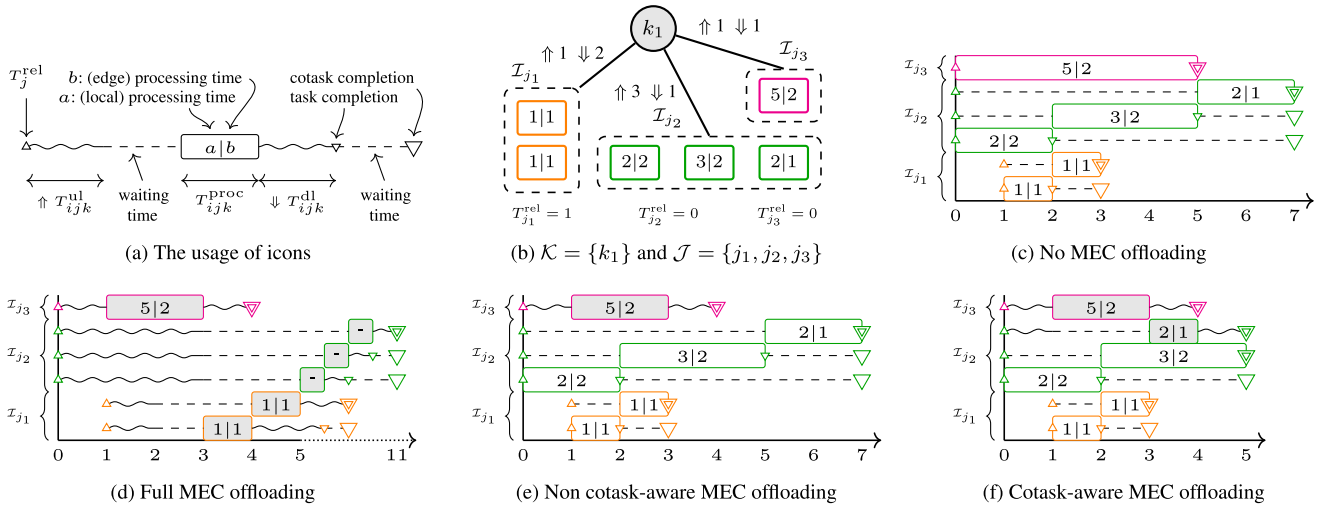


FIGURE 2. A motivating example for MEC offloading (ACCT = 5.00, 7.33, 4.67 and 4.00, respectively).

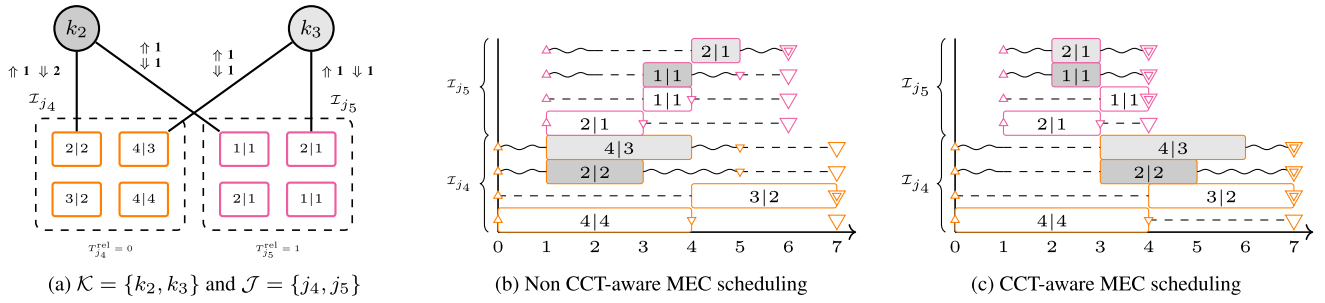


FIGURE 3. A motivating example for MEC scheduling (ACCT = 6.5 and 5.5, respectively).

where $\mathcal{L}_k = \left\{j \middle| \sum_{i\in\mathcal{I}_j} z_{ijk} > 0\right\}$ refers to the set of cotasks offloaded to ES k. For instance, $\sigma_k\left(j'\right) = c$ indicates that j' is the c-th cotask to be processed by ES k.

Since TCT and CCT are our primary performance measures, we define the following two completion time vectors (used to substitute $\boldsymbol{\sigma}$) as

$$\begin{aligned}\text{(task completion)}\ \boldsymbol{\tau} &= \left\{\tau_{jk},\ \ \forall k \in \mathcal{K}_j, j \in \mathcal{J}\right\},\\ \text{(cotask completion)}\ \boldsymbol{t} &= \left\{t_j,\ \ \forall j \in \mathcal{J}\right\},\end{aligned}$$

where τ_{jk} is the TCT of cotask j via PU k and t_j is the CCT of the cotask.

B. CONSTRAINTS

Since each constituent task must be processed by exactly one PU, we have

$$\sum_{k\in\mathcal{K}_j} z_{ijk} = 1, \quad \forall i \in \mathcal{I}_j, j \in \mathcal{J}. \tag{2}$$

After the release of cotask j, the constituent tasks will be processed by various PUs. At each PU, the TCT of the offloaded tasks can be obtained as

$$\tau_{jk} \geq T_j^{\text{rel}} + \Delta_{jk}\left(\boldsymbol{z}\right) + \sum_{i\in\mathcal{I}_j} T_{ijk}^{\text{proc}} z_{ijk}, \quad \forall k \in \mathcal{K}_j, j \in \mathcal{J}, \tag{3}$$

where $\Delta_{jk}\left(\boldsymbol{z}\right) = \max_{i\in\mathcal{I}_j}\left(T_{ijk}^{\text{ul}} + T_{ijk}^{\text{dl}}\right) z_{ijk}$. Since the c-th scheduled cotask is finished earlier than the $(c+1)$-th one on each ES k, we get

$$\tau_{\sigma_k^{-1}(c)k} \leq \tau_{\sigma_k^{-1}(c+1)k}, \quad \forall c \in \{1, 2, \ldots, |\mathcal{L}_k|\},\ k \in \mathcal{K}, \tag{4}$$

where σ_k^{-1} is the inverse function of σ_k and c is a positive ordering index. Recall that each cotask j is completed only if all its constituent tasks are finished and returned to MD Ξ_j. Therefore, the CCT is determined by

$$t_j = \max_{k'\in\mathcal{K}_j} \tau_{jk'} \geq \tau_{jk}, \quad \forall k \in \mathcal{K}_j, j \in \mathcal{J}. \tag{5}$$

In addition, each ES has to produce a schedule to determine the order in which its received tasks are processed, where the resulting TCT can be characterized by the scheduling polyhedron [20] as follows:

$$\begin{aligned}&\sum_{j\in\mathcal{J}'}\left[\tau_{jk} - T_j^{\text{rel}} - \Delta_{jk}\left(\boldsymbol{z}\right)\right]\sum_{i\in\mathcal{I}_j} T_{ijk}^{\text{proc}} z_{ijk} \\ &\geq \frac{1}{2}\left(\sum_{j\in\mathcal{J}'}\sum_{i\in\mathcal{I}_j} T_{ijk}^{\text{proc}} z_{ijk}\right)^2 + \frac{1}{2}\sum_{j\in\mathcal{J}'}\left(\sum_{i\in\mathcal{I}_j} T_{ijk}^{\text{proc}} z_{ijk}\right)^2, \\ &\forall \mathcal{J}' \subseteq \mathcal{J},\ k \in \mathcal{K}.\end{aligned} \tag{6}$$

The inequalities (6) describe the convex hull of feasible completion time vectors of cotasks in the absence of precedence constraints for each individual ES, and they are also known as *parallel inequalities*. Since each MD simply processes its own tasks in our system, (6) naturally holds for all MDs, and hence we only need to make it valid for all ESs.

C. OPTIMIZATION PROGRAMS

The goal of our optimization is to design cotask-aware MEC offloading and scheduling so that average CCT over all cotasks can be minimized. Mathematically, we formulate the problem of joint **co**task-aware **o**ffloading and schedu**l**ing for M**E**C systems (Cool-Edge) as

$$(\mathbb{P}_1)\ \min_{\boldsymbol{\sigma},z}\ \mathsf{ACCT}\left\langle \boldsymbol{\sigma}, z\right\rangle \triangleq \frac{1}{|\mathcal{J}|}\sum_{j\in\mathcal{J}} t_j,$$
$$\text{s.t. (2), (3), (4), (5), (6),}$$
$$z_{ijk}\in\{0,1\},\quad \forall i\in\mathcal{I}_j,\ k\in\mathcal{K}_j,\ j\in\mathcal{J}. \tag{7}$$

Since the right-hand sides of (3)-(6) are all non-negative, we do not have to add the auxiliary constraints of $\tau_{jk}\geq 0$ and $t_j\geq 0, \forall k\in\mathcal{K}_j, j\in\mathcal{J}$.

Remark 1: The spanned subsets $\mathcal{J}'\subseteq\mathcal{J}$ have up to $2^{|\mathcal{J}|}$ combination, which makes the Cool-Edge problem even more computationally prohibitive. Hence, it is desirable to design algorithms without scanning through all possible $\mathcal{J}'$.

Theorem 1: The Cool-Edge problem is NP-hard.

Proof: The NP-hardness of the Cool-Edge problem can be proved by restriction. In particular, we consider an MEC system where PUs are identical and only one cotask j° is awaiting for processing. We describe the problem instance as

$$\mathsf{CE}\left(\left\{\mathcal{I}_{j^\circ}\right\},\mathcal{K},\left\{T^{\text{rel}}_{j^\circ}\right\},\left\{T^{\text{ul}}_{ij^\circ k}\right\},\left\{T^{\text{dl}}_{ij^\circ k}\right\},\left\{T^{\text{proc}}_{ij^\circ k}\right\},X\right).$$

Consider the problem of multi-processor scheduling (MPS) [21] as follows:

- INSTANCE: a set $\mathcal{M}$ of processors, a set $\mathcal{N}$ of tasks, a length $L_n\in Z^+, \forall n\in\mathcal{N}$, and a deadline $Y\in Z^+$.
- QUESTION: Is there an $|\mathcal{M}|$-processor schedule for $\mathcal{N}$ that meets the deadline Y?

Given an MPS problem instance $\mathsf{MPS}\left(\mathcal{M},\mathcal{N},\{L_n\},Y\right)$, we have the corresponding restricted Cool-Edge problem instance as $\mathsf{CE}\left(\mathcal{N},\mathcal{M},\mathbf{0},\mathbf{0},\mathbf{0},\{L_n\},Y\right)$, where $\mathbf{0}$ represents a set of zeros. Since the transformation from each MPS problem instance to the corresponding restricted Cool-Edge problem instance is a one-to-one mapping, which can be run in polynomial time, the proof completes. ■

It can be seen from $\mathbb{P}_1$ that the objective function and most of the constraints are interpreted by completion time vectors. Therefore, we can get rid of $\boldsymbol{\sigma}$ and obtain a relaxed optimization problem as

$$(\mathbb{P}_2)\ \min_{\boldsymbol{t},z}\ \mathsf{ACCT}'\left\langle \boldsymbol{t}, z\right\rangle \triangleq \frac{1}{|\mathcal{J}|}\sum_{j\in\mathcal{J}} t_j,$$
$$\text{s.t. } t_j\geq T^{\text{rel}}_j+\Delta_{jk}(z)+F_{jk}(z),\quad \forall k\in\mathcal{K}_j,\ j\in\mathcal{J}, \tag{8}$$
$$\sum_{j\in\mathcal{J}'} t_j F_{jk}(z)\geq G_{k\mathcal{J}'}(z),\quad \forall\mathcal{J}'\subseteq\mathcal{J},\ k\in\mathcal{K}, \tag{9}$$
$$\sum_{k\in\mathcal{K}_j} z_{ijk}=1,\quad \forall i\in\mathcal{I}_j,\ j\in\mathcal{J}, \tag{10}$$
$$z_{ijk}\in\{0,1\},\quad \forall i\in\mathcal{I}_j,\ k\in\mathcal{K}_j,\ j\in\mathcal{J}, \tag{11}$$

where F_{jk} and $G_{k\mathcal{J}'}(z)$ are expressed, respectively, as

$$F_{jk}(z)=\sum_{i\in\mathcal{I}_j} T^{\text{proc}}_{ijk} z_{ijk}, \tag{12}$$
$$G_{k\mathcal{J}'}(z)=\frac{1}{2}\left[\left(\sum\nolimits_{j\in\mathcal{J}'} F_{jk}(z)\right)^2+\sum\nolimits_{j\in\mathcal{J}'} F^2_{jk}(z)\right]+\sum_{j\in\mathcal{J}'}\left[T^{\text{rel}}_j+\Delta_{jk}(z)\right]F_{jk}(z). \tag{13}$$

Evidently, $\mathbb{P}_2$ does not produce exact schedules $\boldsymbol{\sigma}$, and hence it serves as a relaxed version of $\mathbb{P}_1$. By showing that $\mathbb{P}_1$ is equivalent to $\mathbb{P}_2$ (see Lemma 1), we will focus on designing an approximate solution for $\mathbb{P}_2$ alternatively.

Lemma 1: $\mathsf{ACCT}\left\langle \boldsymbol{\sigma}^*, z^*\right\rangle=\mathsf{ACCT}'\left\langle \boldsymbol{t}^*, z^*\right\rangle$.

Proof: By substituting (5) into (3) and (6), it is apparent that $\mathbb{P}_1$ and $\mathbb{P}_2$ share identical feasible solutions. As $\mathbb{P}_2$ has the same objective function, it retains the optimality of $\mathbb{P}_1$. ■

V. ALGORITHM DESIGN AND ANALYSIS

In the Cool-Edge problem, MEC offloading determines whether to process tasks on the edge and MEC scheduling produces schedules for the offloaded tasks. In the following, we will design the LPR-based cotask offloading algorithm and the ECAF-based cotask scheduling algorithm, and prove the approximation factor jointly achieved by the two algorithms.

A. COTASK-AWARE OFFLOADING DESIGN

Whenever a cotask j is generated, MD Ξ_j will send a set of time information of the cotask to the controller. The controller will make an offloading decision (in Algorithm 1) by solving the following LP, apply the LPR technique based on its latest schedule, and then send the decision back to the hosting MD for offloading. We define the LP as

$$(\mathbb{P}_3)\ \min_{\lambda_j}\ \lambda_j$$
$$\text{s.t. }\sum\nolimits_{i:(i,k)\in\mathcal{R}_j(\lambda_j)} T^{\text{proc}}_{ijk}\zeta_{ijk}\leq\lambda_j,\quad \forall k\in\mathcal{K}_j, \tag{14}$$
$$\sum\nolimits_{k:(i,k)\in\mathcal{R}_j(\lambda_j)}\zeta_{ijk}=1,\quad \forall i\in\mathcal{I}_j, \tag{15}$$
$$\zeta_{ijk}\begin{cases}\geq 0, & \text{if } (i,k)\in\mathcal{R}_j(\lambda_j),\\ =0, & \text{otherwise,}\end{cases} \tag{16}$$
$$\mathcal{R}_j(\lambda_j)=\left\{(i,k)\,\middle|\,T^{\text{proc}}_{ijk}\leq\lambda_j, i\in\mathcal{I}_j, k\in\mathcal{K}_j\right\}, \tag{17}$$

where (14) indicates that the makespan of the task assignment is at most λ_j, (15) ensures that each task is assigned exactly once, (16) and (17) are the auxiliary constraints.

Algorithm 1 The LPR-Based Cotask-Aware Offloading

Input: $\{\mathcal{I}_j\}, \mathcal{J}, \{\mathcal{K}_j\}, \left\{T_{ijk}^{\mathrm{ul}}\right\}, \left\{T_{ijk}^{\mathrm{proc}}\right\}, \left\{T_{ijk}^{\mathrm{dl}}\right\}, \delta, \boldsymbol{\sigma}^{\dagger}$.
Output: $z^{\dagger}$.

procedure CoOFLD
  for $j \in \mathcal{J}$ **do**
    Initialize $\boldsymbol{\zeta}_j' \leftarrow \mathbf{0}$, $\boldsymbol{\zeta}_j'' \leftarrow \mathbf{0}$ and $\boldsymbol{z}_j^{\dagger} \leftarrow \mathbf{0}$;
    for $i \in \mathcal{I}_j$ **do** ▷ Initialization
      Set $k^{\circ} \leftarrow \arg\min_{k \in \mathcal{K}_j} T_{ijk}^{\mathrm{proc}}$ and $\zeta_{ijk^{\circ}}' \leftarrow 1$;
    Run Min-Makespan to obtain $\lambda_j^{\star}$;
    Construct $\mathcal{G}_j\left(\mathcal{I}_j^2, \mathcal{K}_j, \mathcal{E}_j^2\right)$; ▷ Bipartite graph
    Find a perfect matching $\bar{\mathcal{E}}_j^2$ from $\mathcal{G}_j$;
    for $(i,k) \in \mathcal{I}_j \times \mathcal{K}_j$ **do** ▷ Rounding
      if $(i,k) \in \mathcal{E}_j^1 \cup \bar{\mathcal{E}}_j^2$ **then**
        Set $\zeta_{ijk}'' \leftarrow 1$.
    Set $\Lambda_j \leftarrow$ ▷ CCT comparison (local v.s edge)
    $\max_{k \in \mathcal{K}_j} \left\{ \Delta_{jk} + F_{jk}\left(\boldsymbol{\zeta}_j''\right), \sum_{j':\sigma^{\dagger}(j') \le \sigma^{\dagger}(j)} F_{j'k}\left(\boldsymbol{\zeta}_j''\right) \right\}$;
    if $\Lambda_j \le \boxdot$ **then**
      Update $\boldsymbol{z}_j^{\dagger} \leftarrow \boldsymbol{\zeta}_j''$;
  Set $z^{\dagger} \leftarrow \left\{\boldsymbol{z}_j^{\dagger}\right\}$.
procedure Min-Makespan
  Set $\lambda_j^{\max} \leftarrow \max_{k \in \mathcal{K}_j} \sum_{i \in \mathcal{I}_j} T_{ijk}^{\mathrm{proc}} \zeta_{ijk}'$ and $\lambda_j^{\min} \leftarrow \lambda_j^{\max} / \left|\mathcal{K}_j\right|$;
  Set $\lambda_j^{\mathrm{left}} \leftarrow \lambda_j^{\min}$ and $\lambda_j^{\mathrm{right}} \leftarrow \lambda_j^{\max}$;
  while $\lambda_j^{\mathrm{right}} - \lambda_j^{\mathrm{left}} > \delta \lambda_j^{\min}$ **do** ▷ Binary search
    Set $\lambda_j^{\mathrm{mid}} \leftarrow \left(\lambda_j^{\mathrm{right}} + \lambda_j^{\mathrm{left}}\right)/2$;
    if $\exists \boldsymbol{\zeta}_j^{\mathrm{LP}}$ that is a feasible extreme point solution to $\mathbb{P}_3\left(\lambda_j^{\mathrm{mid}}, \mathcal{R}_j\left(\lambda_j^{\mathrm{mid}}\right)\right)$ **then**
      Set $\lambda_j^{\star} \leftarrow \lambda_j^{\mathrm{mid}}$ and update $\lambda_j^{\mathrm{right}} \leftarrow \lambda_j^{\mathrm{mid}}$;
    else
      Update $\lambda_j^{\mathrm{left}} \leftarrow \lambda_j^{\mathrm{mid}}$;

The objective of $\mathbb{P}_3$ is to find a task assignment $\boldsymbol{\zeta}_j = \left\{\zeta_{ijk} | i \in \mathcal{I}_j, k \in \mathcal{K}_j\right\}$ that gives the minimum makespan $\lambda_j^{\star}$. Initially, each constituent task of cotask j is assigned to the PU on which it has the shortest processing time. Then, we use a binary search to find out an optimal value $\lambda_j^{\star}$ for $\mathbb{P}_3$. Since the interval of the binary search in general is composed of non-negative real numbers, we use $\delta > 0$ as an error-tolerant parameter to ensure that the binary search (between the left and right boundaries $\lambda_j^{\min}$ and $\lambda_j^{\max}$, respectively) terminates in polynomial time. On the convergence of the binary search, we are given with $\boldsymbol{\zeta}_j^{\mathrm{LP}}$ for cotask j, based on which we construct the sets of task-PU pairs $\mathcal{E}_j^{(\cdot)} = \{(i,k) | i \in \mathcal{I}_j^{(\cdot)}, k \in \mathcal{K}_j\}$, for $(\cdot) \in \{1,2\}$, where $\mathcal{I}_j^1 = \{i | \exists k \in \mathcal{K}_j \text{ s.t. } \zeta_{ijk}^{\mathrm{LP}} = 1\}$ and $\mathcal{I}_j^2 = \{i | \exists k \in \mathcal{K}_j \text{ s.t. } 0 < \zeta_{ijk}^{\mathrm{LP}} < 1\}$. Since $\mathcal{E}_j^1$ includes the task-PU pairs that are with integral values, it can be output directly. On the other hand, $\mathcal{E}_j^2$ collects the task-PU pairs that are with fractional values, so we need to obtain rounded values from it. To this end, we construct a bipartite graph $\mathcal{G}_j$ that possesses a perfect matching (the existence is asserted by [22]), and then use the Hungarian Method [23] to output all task-PU pairs that appear in the matching.

Algorithm 2 The ECAF-Based Cotask-Aware Scheduling

Input: $\{\mathcal{I}_j\}, \mathcal{J}, \mathcal{K}, \left\{T_j^{\mathrm{rel}}\right\}, \left\{T_{ijk}^{\mathrm{ul}}\right\}, \left\{T_{ijk}^{\mathrm{proc}}\right\}$.
Output: $\boldsymbol{\sigma}^{\dagger}$.

procedure CoSKED
  Initialize $\sigma_k^{\dagger}(j) \leftarrow 0, \forall j \in \mathcal{J}, k \in \mathcal{K}$;
  Set the set $\mathcal{J}^{\circ}$ of unscheduled cotasks to $\mathcal{J}$;
  for $c = 1$ to $|\mathcal{J}|$ **do**
    Obtain the next earliest arriving cotask $j^{\circ} \leftarrow \arg\min_{j \in \mathcal{J}^{\circ}} \left(T_j^{\mathrm{rel}} + \min_{i \in \mathcal{I}_j, k \in \mathcal{K}} T_{ijk}^{\mathrm{ul}}\right)$;
    for $k \in \mathcal{K}$ **do**
      Set $\sigma_k^{\dagger}(j^{\circ}) \leftarrow c$;
    Update $\mathcal{J}^{\circ} \leftarrow \mathcal{J}^{\circ} \setminus \{j^{\circ}\}$;
  Set $\boldsymbol{\sigma}^{\dagger} \leftarrow \left\{\sigma_k^{\dagger}\right\}$.

B. COTASK-AWARE SCHEDULING DESIGN

Given a set of generated cotasks, the controller will produce the function $\boldsymbol{\sigma}^{\dagger}$ (as the output of Algorithm 2) to determine the ordering of processing those cotasks. The controller updates the schedule based on the ECAF rule according to the collected time information: the cotask that arrives at any ES earliest is scheduled first. Whenever the schedule is updated, the controller promptly informs ESs with the latest one. Note that the controller keeps exactly one schedule that is followed by all ESs, and hence cotasks are to be processed synchronously among ESs.

Remark 2: The ECAF rule can be viewed as a variant of the first-come-first-served (FCFS) rule for cotasks, as it produces schedules based on the entirety of each cotask, instead of the constituent tasks. In addition, the produced schedules are applied to all PUs, instead of individual ones.

C. APPROXIMATION FACTOR

In the following, we prove that the achieved ACCT can be upper-bounded, thereby obtaining the approximation factor of our proposed solution.

Definition 1: The ratio of machine computability, the error-tolerant parameter and the ratio of radio access delays are defined, respectively, as

$$\begin{aligned}
&\textit{(machine computability)} && \kappa := \max_{i \in \mathcal{I}_j, j \in \mathcal{J}} \left(T_{ij\Xi_j}^{\mathrm{proc}} / \min_{k \in \mathcal{K}} T_{ijk}^{\mathrm{proc}}\right), \\
&\textit{(error tolerance)} && \epsilon := \delta \max_{j \in \mathcal{J}} \left(\lambda_j^{\min} / \lambda_j^{\max}\right) / (1 - \delta), \\
&\textit{(radio access delays)} && \rho := \max_{j \in \mathcal{J}} \left(\frac{\max_{k \in \mathcal{K}} T_{ijk}^{\mathrm{ul}}}{\min_{k \in \mathcal{K}} T_{ijk}^{\mathrm{ul}}}, \frac{\max_{k \in \mathcal{K}} T_{ijk}^{\mathrm{dl}}}{\min_{k \in \mathcal{K}} T_{ijk}^{\mathrm{dl}}} \right).
\end{aligned}$$

Lemma 2: For any $\epsilon > 0$, $\mathsf{ACCT}'\left\langle \boldsymbol{t}^\dagger, z^\dagger \right\rangle \leq \Psi \mathsf{ACCT}'\left\langle \boldsymbol{t}^*, z^* \right\rangle$, *where* $\Psi = 4\kappa\,(1+\epsilon) + \rho + 1$.

Proof: According to CoSKED, all ESs adopt the same schedule. Let j_c be the c-th scheduled cotask. Then, the achieved ACCT can be expressed as

$$\mathsf{ACCT}'\left\langle \boldsymbol{t}^\dagger, z^\dagger \right\rangle = \frac{1}{|\mathcal{J}|}\sum_{j\in\mathcal{J}} t_j^\dagger = \frac{1}{|\mathcal{J}|}\sum_{c=1}^{|\mathcal{J}|} t_{j_c}^\dagger. \tag{18}$$

The ECAF rule reveals that j_c is the c-th earliest arriving cotask, only the set $\mathcal{J}_c = \{j_1, j_2, \ldots, j_c\}$ of cotasks precedes j_c. Therefore, the following upper bound holds for j_c:

$$t_{j_c}^\dagger \leq \min\left\{ \overbrace{T_{j_c}^{\mathrm{rel}} + \Delta_{j_c j_c} + \sum_{i\in\mathcal{I}_{j_c}} T_{ij_c\Xi_{j_c}}^{\mathrm{proc}}}^{\text{the elapsed time to finish processing } j_c \text{ locally } (=\boxdot)}, \right.$$
$$\left. \overbrace{\max_{k\in\mathcal{K}_{j_c}}\left[\underbrace{\max_{1\leq d\leq c}\left(T_{j_d}^{\mathrm{rel}} + T_{j_d k}^{\mathrm{ul}}\right)}_{\text{all the cotasks preceding } j_c \text{ have been released after this point of time } (=\boxminus)} + \sum_{d=1}^{c} F_{j_d k}\left(z^\dagger\right) + T_{j_c k}^{\mathrm{dl}} \right]}^{\text{the maximum elapsed time to finish processing } j_c\ (=\boxplus)} \right\}. \tag{19}$$

By substituting (19) into (18), we get

$$\mathsf{ACCT}'\left\langle \boldsymbol{t}^\dagger, z^\dagger \right\rangle \leq \frac{1}{|\mathcal{J}|}\sum_{c=1}^{|\mathcal{J}|} \min\left\{\boxdot, \boxminus + \boxplus + \boxtimes\right\}, \tag{20}$$

where $\boxtimes = \max_{k\in\mathcal{K}_{j_c}} T_{j_c k}^{\mathrm{dl}}$. According to CoOFLD, since each MD offloads tasks only if the offloading decision suggested by the controller provides a lower ACCT, we have

$$\min\left\{\boxdot, \boxminus + \boxplus + \boxtimes\right\} = \boxminus + \boxplus + \boxtimes. \tag{21}$$

In the following, we show how to upper-bound $\boxminus$, $\boxplus$ and $\boxtimes$.

$\boxminus$ refers to the latest arrival time (on a PU) of the offloaded tasks of j_c. By the definition of ρ, we get

$$\begin{aligned}\boxminus &\leq \max_{1\leq d\leq c}\left(T_{j_d}^{\mathrm{rel}} + \max_{k\in\mathcal{K}_{j_c}} T_{j_d k}^{\mathrm{ul}}\right)\\ &\leq \rho \min_{k\in\mathcal{K}} \max_{1\leq d\leq c}\left(T_{j_d}^{\mathrm{rel}} + T_{j_d k}^{\mathrm{ul}}\right) \leq \rho t_{j_c}^*,\end{aligned} \tag{22}$$

where the last inequality follows from the ECAF rule. According to (8), $\boxtimes \leq t_{j_c}^*$. Therefore,

$$\frac{1}{|\mathcal{J}|}\sum_{c=1}^{|\mathcal{J}|}(\boxminus + \boxtimes) \leq (\rho+1)\,\mathsf{ACCT}'\left\langle \boldsymbol{t}^*, z^* \right\rangle. \tag{23}$$

$\boxplus$ refers to the effective processing time of j_c, which is determined by the makespan sum of the preceding cotasks and the net local processing time. If we look at the makespan sum, coping with the makespan of each cotask is analogous to *scheduling unrelated parallel machines*.[5] By applying the LPR technique, we see (in Appendix A) that

$$\boxplus \leq \sum_{d=1}^{c} \max_{k\in\mathcal{K}_{j_c}} F_{j_d k}\left(z^\dagger\right) \leq 2\kappa\,(1+\epsilon)\cdot\circledast, \tag{24}$$

where $\circledast = \max_{k\in\mathcal{K}_{j_c}} \sum_{d=1}^{c} F_{j_d k_d}\left(z_{j_d}^*\right)$. Since $\mathbb{P}_2$ given with z^* is a purely an LP, we further use its dual LP to obtain the upper bound (see Appendix B) as

$$\frac{1}{|\mathcal{J}|}\sum_{c=1}^{|\mathcal{J}|}\circledast \leq \left(\frac{2\,|\mathcal{J}|}{|\mathcal{J}|+1}\right)\mathsf{ACCT}'\left\langle \boldsymbol{t}^*, z^* \right\rangle. \tag{25}$$

By applying (21)-(25) to (20), we complete the proof. ■

Our proposed solution is with a polynomial-time complexity. By the facts that converging the binary search, solving $\mathbb{P}_3$ (via the Karmarkar's algorithm [25]) and finding a perfect matching (by the Hungarian method [23]) are with the time complexity of $\mathcal{O}\left(\log\frac{1}{\epsilon}\right)$, $\mathcal{O}\left(|\mathcal{I}|^{3.5}\,|\mathcal{K}|^{3.5}\,L^2\right)$ and $\mathcal{O}\left((|\mathcal{I}|+|\mathcal{K}|)^3\right)$, respectively, where L is the number of bits used to represent each numeric input in unary. Overall, we have the time complexity of $\mathcal{O}\left(|\mathcal{I}|^{3.5}\,|\mathcal{K}|^{3.5}\,L^2\log\frac{1}{\epsilon}\right)$.

Using Lemmas 1 and 2 gives rise to the following result.

Theorem 2: For any $\epsilon > 0$, *the proposed solution achieves the approximation factor of* $[4\kappa\,(1+\epsilon) + 2\rho]$ *to the CoolEdge problem.*

TABLE 2. The comparison schemes (Row 1: Acronyms / Row 2: Offloading schemes / Row 3: Scheduling schemes).

MC	RC	CoCo	CS	CL	NN
MT	RR	CoOFLD	CoOFLD	CoOFLD	∅
CoSKED	CoSKED	CoSKED	SPT	LPT	∅

MT: Each task is processed by the PU that minimizes its TCT.
RR: Each MD offloads tasks in a round robin fashion.
SPT: Each ES schedules the shortest-processing-time task first.
LPT: Each ES schedules the longest-processing-time task first.
∅: No MDs offload tasks and hence ESs have nothing to schedule.

VI. PERFORMANCE EVALUATION

In this section, we provide testbed experiments and simulations to demonstrate the achieved ACCT performance.

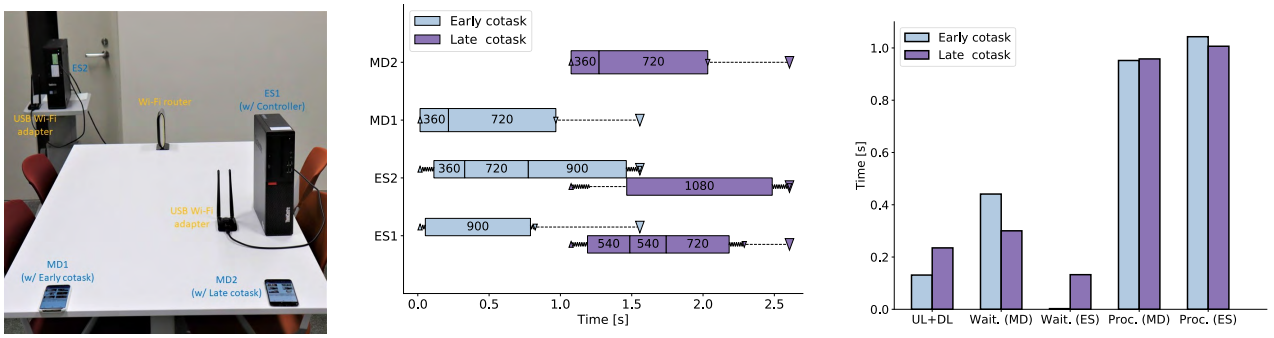
To demonstrate the ACCT performance of our proposed solution, we consider various offloading and scheduling schemes for comparison (see TABLE 2), including

- **CoCo**: the proposed CoOFLD and CoSKED,
- **MC/RC**: existing offloading schemes with CoSKED,
- **CS/CL**: CoOFLD with existing scheduling schemes, and
- **NN**: the baseline scheme (i.e., merely local computation).

A. TESTBED EXPERIMENTS

We construct an MEC system (see FIGURE 4a) of 2 ESs (desktop computers with Intel Core i7-6700K processors) and 2 MDs (smartphones with the Qualcomm Snapdragon 835 mobile PC platform), where each ES is equipped with

[5] By using the three-field classification in the scheduling theory [24], this problem is conventionally denoted as $R_m||C_{\max}$.



(a) Environment settings (b) The processing time of the two cotasks (c) The elapsed time profile

FIGURE 4. Testbed environment and the measurement results of the elapsed time under CoCo.

TABLE 3. The elapsed time for processing (Row 1: Image quality / Rows 2 & 3: ES's & MD's Processing time / Row 4: Machine computability ratio).

640×360	960×540	1280×720	1600×900	1920×1080
0.1145 [s]	0.2556 [s]	0.4446 [s]	0.6966 [s]	0.9889 [s]
0.1893 [s]	0.4319 [s]	0.7572 [s]	1.1863 [s]	1.7087 [s]
1.653	1.689	1.703	1.703	1.728

a USB Wi-Fi adapter (Alfa Network AWUS036ACH). The controller is implemented in one of the ESs, and it connects ESs through a Wi-Fi router (Aterm WG2600HP2).

A simple application is made to capture the cotask feature in the MEC system: the mission of each cotask is to count the detected frontal faces, and the constituent tasks represent a set of images for counting. To this end, we collect 10^3 images from Pexels [26] and use Dlib [27] to detect frontal faces. The set of collected images consists of five different image qualities. We measure the elapsed processing time (see TABLE 3), which is shown to grow with image qualities. These measurement results will further be used to set the processing time of tasks as well as the ratio of machine computability in our simulations.

In our testbed (see FIGURE 4a), we randomly choose 6 images to form a cotask. FIGURE 4b (wherein the icons follow the usage in FIGURE 2a) illustrates the spatial-temporal dynamics of the two cotasks. It can be seen that CoOFLD produces a balanced offloading while CoSKED enables the early cotask to be scheduled first, which conforms to our designed solution. FIGURE 4c indicates the corresponding elapsed time on average, from which we see that all PUs are efficiently utilized and no much waiting time on ESs, which affirm the practicability of our proposed solution. Moreover, we observe that the effect of shared network bandwidth (in terms of uplink and downlink delays) is not pronounced, since the effect of shared computing resources (in terms of processing and waiting time) plays a more dominant role in the resulting CCT.

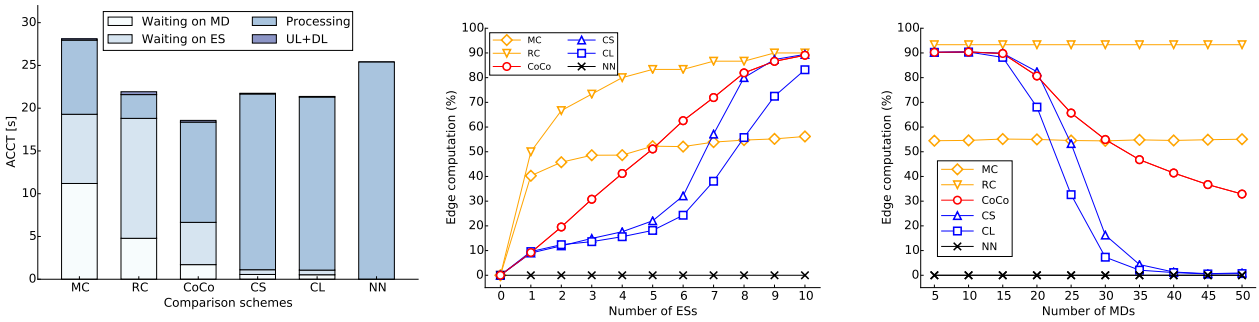
B. SIMULATIONS

Here, we consider an MEC system of 5 ESs and 15 MDs. Each cotask is formed by randomly selected 30 images, and the release time of cotasks is determined by a Poisson process with the mean of 0.1 seconds. The uplink and downlink delays between an MD and an ES are both uniformly distributed in the range of $[0.1, 0.3]$ seconds. The error-tolerant parameter is set to $1/10^8$. According to the measurement results in TABLE 3, we set the ratio of machine computability to 1.7 unless stated otherwise. All results are averaged over 100 iterations.

Our proposed solution is shown to outperform the comparison schemes in terms of ACCT. In FIGURE 5a, we see that CoCo strikes a good balance between processing time and waiting time. Specifically, CoCo gains from lower waiting time as compared with MC and RC, since it intends to offload tasks to PUs in a balanced way to avoid stacking tasks on a PU. On the other hand, CoCo enables cotasks to be scheduled in the same order on all ESs, and hence it prevents the early completed tasks of a cotask from waiting for the completion of other constituent tasks, thereby achieving lower ACCT with respect to CL, CS and NN. Altogether, CoCo offloads tasks in a balanced way and schedule cotasks synchronously, and therefore it performs better than the comparison schemes.

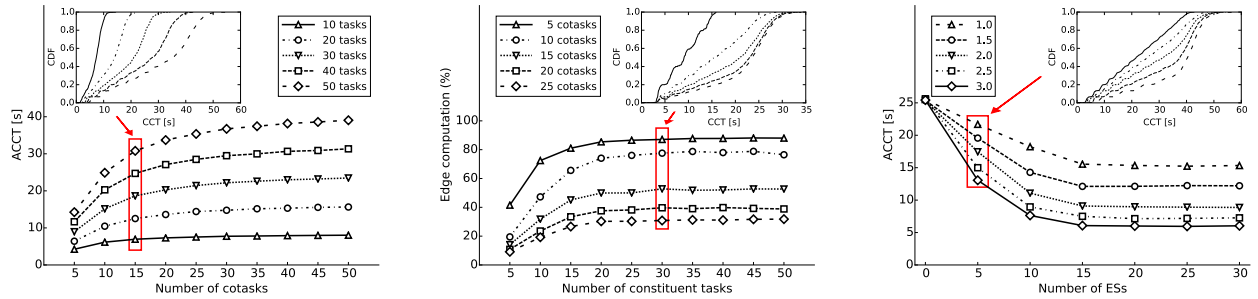
The exploitation of edge computation provides evidences to the achieved ACCT gain. FIGURE 5b and 5c show that the exploitation of edge computation increases with the number of ESs and decreases with that of MDs, respectively. This is because the benefit of edge computation is pronounced if there are not too many tasks queued on ESs. If the number of MDs are relatively larger than that of ESs, the benefit of edge computation diminishes since the offloaded tasks will encounter large waiting time, and hence less edge computation is preferable. Another observation is that the exploitation of edge computation essentially does not approach to 100%. The reason is that each hosting MD needs to wait for the last returned task. If the hosting MD can process a small portion of its tasks locally while waiting for the offloaded tasks to return, its CCT can be further reduced as compared with doing all computation on the edge.

The numbers of constituent tasks, cotasks and ESs have prompt effects on ACCT performance. In FIGURE 6a and 6b, we see that ACCT increases with the numbers of constituent tasks and cotasks. This is because there are more tasks to be



(a) The elapsed time profile (b) The impact of the number of ESs (c) The impact of the number of MDs

FIGURE 5. The achieved ACCT and the exploitation of edge computation among the comparison schemes.



(a) The impact of the number of cotasks (b) The impact of the number of constituent tasks (c) The impact of the number of ESs

FIGURE 6. The achieved ACCT and the exploitation of edge computation under CoCo.

processed, and hence longer elapsed time for processing can be observed. In addition, as the number of tasks per cotask increases, the exploitation of edge computation first increases and then saturates. The saturation happens since plenty of tasks are waiting for processing on ESs, from which we see that a balanced local and edge computation attains better ACCT.

The more the number of ESs or the higher ratio of machine computability, the lower the achieved ACCT. In FIGURE 6c, it can be seen that ACCT is lower whenever there are abundant computing resources. Another notable fact is the distribution of computing resources. Given a fixed amount of computing resources, it is better to deploy fewer but more powerful ESs. This is because the offloaded tasks may not be perfectly uniform among PUs since they cannot be divided into fractional pieces. Therefore, from the perspective of PUs, fewer PUs gives rise to more uniform offloading and lower ACCT.

VII. CONCLUSION

The presence of cotasks necessitates a joint design of MEC offloading and scheduling to better utilize the networked computing resources. In this paper, we formulate the CoolEdge problem as an MINLP and prove its NP-hardness. We propose the cotask-aware offloading and scheduling algorithms based on the LPR technique and the ECAF rule, respectively. We prove the approximation factor of our proposed solution, which is shown to hinge on the ratios of machine computability and radio access delays. We conduct testbed experiments to show the practicability of our proposed solution. In larger network scales, our simulation results demonstrate that the proposed solution reduces ACCT effectively as cotasks can be offloaded in a balanced way and scheduled synchronously, thereby achieving a better tradeoff between local and edge computation. In addition, the achieved ACCT gain is more pronounced as the number of constituent tasks of each cotask goes smaller or the computing resources of ESs are more abundant or put together.

APPENDIX A

According to the task-PU pairs in $\mathcal{E}_j^1$, each PU can receive at most one task, and hence the partial makespan for $\mathcal{E}_j^1$ can be upper-bounded as

$$(\diamond) := \max_{k \in \mathcal{K}_j} \sum_{i:(i,k) \in \mathcal{E}_j^1} T_{ijk}^{\text{proc}} z_{ijk}^{\dagger} \leq \max_{(i,k) \in \mathcal{E}_j^1} T_{ijk}^{\text{proc}} \leq \lambda_j^{\star}. \quad (26)$$

On the other hand, we recall that, for each cotask j, the graph $\mathcal{G}_j$ has a perfect matching. As a consequence, a set of disjoint task-PU pairs is obtainable from $\tilde{\mathcal{I}}_j^2$ and hence each PU receives at most one task. Similarly, we have

$$(\diamond\diamond) := \max_{k \in \mathcal{K}_j} \sum_{i \in \tilde{\mathcal{E}}_j^2} T_{ijk}^{\text{proc}} z_{ijk}^{\dagger} \leq \lambda_j^{\star}. \quad (27)$$

Since the binary search in CoOFLD terminates in the interval of $\left[\lambda_j^\star - \delta\lambda_j^{\min}, \lambda_j^\star\right]$, we have

$$\lambda_j^\star \le (1+\epsilon)\,\lambda_j^{\mathrm{LP}}. \tag{28}$$

With (26) and (27), we conclude that

$$\max_{k\in\mathcal{K}_j} F_{jk}\left(\boldsymbol{z}_j^\dagger\right) = (\diamond) + (\diamond\diamond) \le 2\,(1+\epsilon)\,\lambda_j^{\mathrm{LP}}. \tag{29}$$

By applying the fact of $\lambda_j^{\mathrm{LP}} \le \max_{k\in\mathcal{K}_j} F_{jk}\left(\boldsymbol{z}_j^*\right)$ to (29), we complete the proof.

APPENDIX B

Since the first term of (20) is upper-bounded by (23), we only have to focus on the second term. Given with $\boldsymbol{z}^*$, we can get rid of (8), (10) and (11), thereby obtaining as

$$\begin{aligned}(\mathbb{P}_4)\quad &\min_{\boldsymbol{t}}\ \mathsf{ACCT}''\left\langle \boldsymbol{t}, \boldsymbol{z}^*\right\rangle \triangleq \frac{1}{|\mathcal{J}|}\sum_{j\in\mathcal{J}} t_j,\\ &\text{s.t.}\quad \sum_{j\in\mathcal{J}'} t_j F_{jk}\left(\boldsymbol{z}^*\right) \ge G_{k\mathcal{J}'}\left(\boldsymbol{z}^*\right), \quad \forall \mathcal{J}'\subseteq\mathcal{J},\ k\in\mathcal{K}.\end{aligned} \tag{30}$$

Then, the corresponding dual LP can be obtained as

$$\begin{aligned}(\mathbb{P}_5)\quad &\max_{\boldsymbol{\gamma}}\ \sum_{k\in\mathcal{K}}\sum_{\mathcal{J}'\in\mathcal{J}} G_{k\mathcal{J}'}\left(\boldsymbol{z}^*\right)\gamma_{k\mathcal{J}'},\\ &\text{s.t.}\quad \sum_{k\in\mathcal{K}} F_{jk}\left(\boldsymbol{z}^*\right)\sum_{\mathcal{J}'\subseteq\mathcal{J}:j\in\mathcal{J}'}\gamma_{k\mathcal{J}'} = \frac{1}{|\mathcal{J}|},\quad \forall j\in\mathcal{J},\end{aligned} \tag{31}$$

$$\gamma_{k\mathcal{J}'} \ge 0,\quad \forall k\in\mathcal{K},\ \mathcal{J}'\subseteq\mathcal{J}. \tag{32}$$

In $\mathbb{P}_5$, we start with an infeasible dual solution by setting all dual variables to zeros, and continue to raise the values to make (31) holds for any of the cotasks. That is, the weighted sum of the dual variables for each cotask j should be equal to $1/|\mathcal{J}|$. Whenever (31) is tightened for a cotask, the corresponding dual variables will be frozen so as to maintain the dual feasibility. This procedure is repeated until (31) becomes tight for all cotasks.

Let us denote by $\boldsymbol{t}^\ddagger$ the ACCT after replacing $\boldsymbol{z}^\dagger$ with $\boldsymbol{z}^*$. According to CoSKED, we associate the dual variables with cotask indexes and set their values through

$$\gamma_{k_c\mathcal{J}'} \leftarrow \frac{1/|\mathcal{J}|}{F_{j_c k_c}(\boldsymbol{z}^*)}\mathbb{1}_{\mathcal{J}'=\mathcal{J}_c}, \tag{33}$$

where $k_c = \arg\max_{k\in\mathcal{K}_{j_c}} \sum_{d=1}^{c} F_{j_d k}\left(\boldsymbol{z}_{j_d}^*\right)$. With (33),

$$\begin{aligned}\mathsf{ACCT}''\left\langle \boldsymbol{t}^\ddagger, \boldsymbol{z}^*\right\rangle &= \sum_{j\in\mathcal{J}}\left(\sum_{k\in\mathcal{K}} F_{jk}\left(\boldsymbol{z}^*\right)\sum_{\mathcal{J}'\subseteq\mathcal{J}:j\in\mathcal{J}'}\gamma_{k\mathcal{J}'}\right) t_j^\ddagger\\ &= \sum_{c=1}^{|\mathcal{J}|}\gamma_{k_c\mathcal{J}_c}\sum_{d=1}^{c} F_{j_d k_c}\left(\boldsymbol{z}^*\right) t_{j_d}^\ddagger\\ &\le \sum_{c=1}^{|\mathcal{J}|}\gamma_{k_c\mathcal{J}_c}\left[\sum_{d=1}^{c} F_{j_d k_c}\left(\boldsymbol{z}^*\right)\right]^2,\end{aligned} \tag{34}$$

where the inequality holds since

$$t_{j_d} \le t_{j_c} = \sum_{d=1}^{c} F_{j_d k_c}\left(\boldsymbol{z}^*\right), \quad \forall d\le c. \tag{35}$$

According to the expression of $G_{k\mathcal{J}_c}(\boldsymbol{z}^*)$, we see that

$$\begin{aligned}&\left[\sum_{d=1}^{c} F_{j_d k_c}\left(\boldsymbol{z}^*\right)\right]^2\\ &\quad\le \sum_{c=1}^{|\mathcal{J}|}\left[2\,G_{k_c\mathcal{J}_c}\left(\boldsymbol{z}^*\right) - \sum_{d=1}^{c} F_{j_d k_c}^2\left(\boldsymbol{z}^*\right)\right]\\ &\quad\le \sum_{c=1}^{|\mathcal{J}|}\left[2\,G_{k_c\mathcal{J}_c}\left(\boldsymbol{z}^*\right) - \frac{1}{|\mathcal{J}|}\left(\sum_{d=1}^{c} F_{j_d k}\left(\boldsymbol{z}^*\right)\right)^2\right],\end{aligned} \tag{36}$$

where the second inequality follows from the root-mean square-arithmetic mean inequality. By applying (36) to (34),

$$\mathsf{ACCT}''\left\langle \boldsymbol{t}^\ddagger, \boldsymbol{z}^*\right\rangle \le \left(\frac{2\,|\mathcal{J}|}{|\mathcal{J}|+1}\right)\sum_{c=1}^{|\mathcal{J}|}\gamma_{k_c\mathcal{J}_c} G_{k_c\mathcal{J}_c}\left(\boldsymbol{z}^*\right). \tag{37}$$

Using the weak duality gives rise to

$$\sum_{c=1}^{|\mathcal{J}|}\gamma_{k_c\mathcal{J}_c} G_{k_c\mathcal{J}_c}\left(\boldsymbol{z}^*\right) \le \mathsf{ACCT}''\left\langle \boldsymbol{t}^*, \boldsymbol{z}^*\right\rangle \le \mathsf{ACCT}'\left\langle \boldsymbol{t}^*, \boldsymbol{z}^*\right\rangle, \tag{38}$$

where the second inequality holds since the optimal solution of $\mathbb{P}_2$ is lower-bounded by that of $\mathbb{P}_4$. Combining (37) and (38) therefore completes the proof.

REFERENCES

[1] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, Sep. 2015.

[2] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust, "Mobile-edge computing architecture: The role of MEC in the Internet of Things," *IEEE Consum. Electron. Mag.*, vol. 5, no. 4, pp. 84–91, Oct. 2016.

[3] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[4] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017.

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

[6] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 12, no. 9, pp. 4569–4581, Sep. 2013.

[7] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. Signal Inf. Process. Over Netw.*, vol. 1, no. 2, pp. 89–103, Jun. 2015.

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

[9] L. Wang, L. Jiao, D. Kliazovich, and P. Bouvry, "Reconciling task assignment and scheduling in mobile edge clouds," in *Proc. IEEE ICNP*, Nov. 2016, pp. 1–6.

[10] H. Tan, Z. Han, X.-Y. Li, and F. C. M. Lau, "Online job dispatching and scheduling in edge-clouds," in *Proc. IEEE INFOCOM*, May 2017, pp. 1–9.

[11] X. Lyu, W. Ni, H. Tian, R. P. Liu, X. Wang, G. B. Giannakis, and A. Paulraj, "Optimal schedule of mobile edge computing for Internet of Things using partial information," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2606–2615, Nov. 2017.
[12] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 945–953.
[13] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *Proc. IEEE INFOCOM Workshops*, Apr. 2014, pp. 352–357.
[14] L. Yang, B. Liu, J. Cao, Y. Sahni, and Z. Wang, "Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds," in *Proc. IEEE CLOUD*, Jun. 2017, pp. 246–253.
[15] S. E. Mahmoodi, R. N. Uma, and K. P. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 301–313, Apr./Jun. 2018.
[16] H. Chang, M. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee, "Scheduling in MapReduce-like systems for fast completion time," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 3074–3082.
[17] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proc. ACM SIGCOMM*, Aug. 2014, pp. 443–454.
[18] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *Proc. ACM SIGCOMM*, Aug. 2015, pp. 393–406.
[19] Y. Zhao, S. Luo, Y. Wang, and S. Wang, "Cotask scheduling in cloud computing," in *Proc. IEEE ICNP*, Oct. 2017, pp. 1–6.
[20] M. Queyranne, "Structure of a simple scheduling polyhedron," *Math. Program.*, vol. 58, nos. 1–3, pp. 263–285, Jan. 1993.
[21] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1990.
[22] J. K. Lenstra, D. B. Shmoys, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," in *Proc. IEEE SFCS*, Oct. 1987, pp. 1–8.
[23] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Res. Logistics Quart.*, vol. 2, nos. 1–2, pp. 83–97, Mar. 1955.
[24] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. New York, NY, USA: Springer, 2016.
[25] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proc. ACM STOC*, 1984, pp. 302–311.
[26] (Jul. 2018). *Pexels*. [Online]. Available: http://www.pexels.com/
[27] (Jul. 2018). *Dlib*. [Online]. Available: http://dlib.net/

YI-HAN CHIANG received the Ph.D. degree from the Graduate Institute of Communication Engineering, National Taiwan University, Taipei, Taiwan, in 2017. He is currently a Project Assistant Professor with the National Institute of Informatics, Tokyo, Japan. His research interests include mobile edge computing, green wireless networking, network optimization, and approximation algorithms.

TIANYU ZHANG received the B.S. degree from the School of Electronics Engineering and Computer Science, Peking University, Beijing, China, in 2017. He is currently pursuing the master's degree with the National Institute of Informatics, Tokyo, Japan. His research interests include mobile edge computing, machine learning, and reinforcement learning.

YUSHENG JI received the B.E., M.E., and D.E. degrees in electrical engineering from The University of Tokyo. She joined the National Center for Science Information Systems, Japan, in 1990. She is currently a Professor with the National Institute of Informatics and with The Graduate University for Advanced Studies. Her research interests include network architecture, resource management, and quality of service provisioning in wired and wireless communication networks. She is/has been an Editor of the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, a Symposium Co-Chair of the IEEE GLOBECOM 2012 and the IEEE GLOBECOM 2014, and a Track Co-Chair of the IEEE VTC 2016-Fall and IEEE VTC 2017-Fall.