Cooperative Service Placement and Scheduling in Edge Clouds: A Deadline-Driven Approach

Yuqing Li, Wenkuan Dai, Xiaoying Gan, *Member, IEEE,* Haiming Jin, *Member, IEEE,* Luoyi Fu, *Member, IEEE,* Huadong Ma, *Member, IEEE,* and Xinbing Wang, *Senior Member, IEEE*

Abstract—Mobile edge computing enables resource-limited edge clouds (ECs) in federation to help each other with resource-hungry yet delay-sensitive service requests. Contrary to common practice, we acknowledge that mobile services are heterogeneous and the limited storage resources of ECs allow only a subset of services to be placed at the same time. This paper presents a jointly optimized design of cooperative placement and scheduling framework, named JCPS, that pursues *social cost minimization* over time while ensuring diverse user demands. Our main contribution is a novel perspective on cost reduction by exploiting the *spatial-temporal diversities* in *workload and resource cost* among federated ECs. To build a practical edge cloud federation system, we have to consider two major challenges: *user deadline preference* and *ECs' strategic behaviors*. We first formulate and solve the problem of spatially strategic optimization without deadline awareness, which is proved $\mathcal{NP}$-hard. By leveraging user deadline tolerance, we develop a Lyapunov-based *deadline-driven* joint cooperative mechanism under the scenario where the workload and resource information of ECs are known for one-shot global cost minimization. The *service priority* imposed by deadline urgency drives time-critical placement and scheduling, which, combined with cooperative control, enables workloads migrated across different times and ECs. Given selfishness of individual ECs, we further design an auction-based cooperative mechanism to elicit *truthful bids* on workload and resource cost. Rigorous theoretical analysis and extensive simulations are performed, validating the efficiency of JCPS in realizing cost reduction and user satisfaction.

Index Terms—Mobile edge computing, joint cooperative placement and scheduling, user deadline preference, ECs' strategic behaviors

1 Introduction

The proliferation of Internet-of-Things with ongoing mega-trend in computing has given rise to mobile edge computing (MEC), a new paradigm which brings cloud computing capabilities to the network edge (e.g., small base stations) [1], [2]. As a result, MEC can offer users pervasive access to powerful computing capability with low latency, underpinning a variety of resource-hungry yet delay-sensitive applications such as mobile gaming and augmented reality [3], [4], [5]. Nonetheless, compared to dedicated datacenters, edge clouds (ECs) are more limited in resources [6]. Although a few works on offloading computation exceeding the EC's capacity to the remote cloud have appeared [7], [8], relying on a single EC greatly restricts MEC performance.

By exploiting edge cloud federation, MEC enables resource-limited ECs to help each other with tasks [9]. Assisted by virtualization techniques, such federation allows the ECs within specific geographic regions to form a shared resource pool, realizing flexible supply of distributed resources [10]. In general, the workload and resource cost of ECs vary widely over time. Such variation often uncorrelated among different ECs can complement one another, offering opportunities for cost reduction via computation offloading [3]. Extensive research has been devoted to cooperative scheduling among federated ECs [6], [11], [12].

However, most of existing solutions rely on a fundamental implicit assumption: ECs can process whatever types of task requests from users, oblivious to the fact that mobile services are heterogeneous and need to be provisioned first when requested [13], [14], [15]. Provisioning a service requires placing the associated data and code at the network edge, thereby allowing tasks requesting the service to be processed [16], [17]. The problem with this assumption is that unlike datacenters, each EC can only supply certain storage resources, suggesting not all services can be placed at the same time. Which services to place determines which type of tasks to process, manifesting itself in the impact on scheduling performance. Combined with spatially uneven workloads, the bottleneck constraints of resources highlight the critical need for cooperative service placement, which is vital to satisfying diverse user demands. Therefore, the power of edge cloud federation could not be fully unleashed, unless the *optimizations of cooperative placement and scheduling are performed jointly*.

To build a practical edge cloud federation system, we have to cope with two major challenges. The first challenge comes from the *joint cooperative control aligned with user deadline preference*. Our study highlights the role of deadline-sensitive applications, which allow for delays in service only if the tasks can be fulfilled before the deadlines. If cooperative control acts as core strategy for spatial load balancing, then surely leveraging deadline tolerance enables workload migrated across different times. Prior work mostly focuses on global cost optimization [11], [12], where user demands especially for those at peak hours are always aggressively migrated to off-peak hours regardless of deadline urgency, making user satisfaction impaired. Additionally, edge cloud federation remains highly unexplored due to the lack of distinction between time-critical and non-time-critical services. Under limited resources, service priority must be at the epicenter of cooperative control to avoid deadline violations for time-critical services [18]. All of these necessitate the design of a judicious mechanism with spatially-temporally optimized control.

- *Y. Li, W. Dai, X. Gan, H. Jin, L. Fu, and X. Wang are with the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: {liyuqing, daiwenkuan119, ganxiaoying, jinhaiming, yiluofu, xwang8}@sjtu.edu.cn.*
- *H. Ma is with the Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, Beijing University of Posts and Telecommunications, Beijing 100876, China. E-mail: mhd@bupt.edu.cn.*

TABLE 1
Main results in this paper

Deadline awareness	ECs' strategic behaviors	Solution	Optimality	Section
×	×	DO-JCPS	$\frac{2\delta}{\delta-1}$-competitive	4
✓	×	D-JCPS	Asymptotic	5.1
✓	✓	AD-JCPS	Asymptotic	5.2

The second challenge is on the *social cost minimization in case of ECs' strategic behaviors*. By exploiting the diversities in workload and resource cost among different ECs, edge cloud federation provides potential cost reductions via joint cooperative control. A large body of research on cost minimization has emerged [4], [6], [11], most of which implicitly assumes a cooperative scenario where the EC workload and resource information is known for global optimization. But as both offloading requester and provider, ECs are often deployed by self-interested individuals (e.g. enterprise owners). They prefer to utilize better and cheaper resources to fulfill tasks, suggesting the necessity of an efficient market to elicit desirable behaviors from ECs for global optimization to be implemented. Whereas as providers, ECs' sharing offloading may deplete remaining resources for their own use sooner than expected [19]. Hence, the over-exploiting behaviors that harm cooperation performance should be prevented, inevitably doubling the difficulty in realizing benefits of edge cloud federation.

To tackle these challenges, we formulate the joint cooperative placement and scheduling models for cases with and without deadline awareness. The long-term social cost minimization is achieved by leveraging spatial-temporal variations of workload and resource cost among federated ECs. For deadline-oblivious case, we design an online combinatorial algorithm to decompose the $\mathcal{NP}$-hard multi-slot problem into multiple single-slot problems solvable through branch-and-bound. For deadline-driven case, the temporal evolution of deadline urgency is captured by priority queues, which, combined with task queues, provide an integration of service priority to decision makings. By harnessing Lyapunov techniques, the one-shot cost minimization problem is derived from deploying the most time-critical services and utilizing resources as much as possible. Given selfishness of ECs, we design an auction-based cooperative mechanism that encourages them to truthfully disclose workload and resource cost, where the winner determination under truthful bidding is proved equivalent to one-shot global optimization. For more clarity, we list the main results in Table 1, and make the key contributions as follows.

- We propose JCPS, a novel online joint cooperative placement and scheduling framework that realizes cost reduction by exploiting spatial-temporal diversities in workload and resource cost among federated ECs. To the best of our knowledge, we're the first to explore the benefits of *edge cloud federation* by taking both *user deadline preference and ECs' strategic behaviors* into account.
- We formulate the problem of spatially strategic optimization without deadline awareness, and design a 2-approximation algorithm and an online algorithm that work together to solve with a competitive ratio of $\frac{2\delta}{\delta-1}$. By leveraging user deadline tolerance, the spatial-temporal optimality of the problem is expected to achieve. The service priority imposed by deadline urgency drives time-critical placement and scheduling, which, combined with edge cloud federation, enable the workload migrated across different times and ECs.
- With deadline awareness, we consider both the scenario where ECs act cooperatively with workload and resource information known for global optimization, and the scenario where ECs select strategies to solely pursue their own interests. For the latter, we design an auction-based mechanism to elicit truthful bids on workload and resource cost from ECs. We theoretically prove that our mechanisms can guarantee cost optimality while bounding computation workloads and deadline violations, even in case of self-interested ECs.

In what follows, we discuss related works in Section 2 and introduce the system model in Section 3. Design and analysis of the joint cooperative mechanisms are presented in Sections 4 and 5. Simulations and conclusions are given in Sections 6 and 7.

2 Motivation and Related Works

Why Edge Cloud Federation. MEC has emerged as a compelling paradigm by providing cloud-like computing capabilities on the network edge [2], [3], [4]. However, the limited resources available to individual ECs (a.k.a. fog [3]) remain the biggest obstacle [6]. There are mainly two lines of efforts to tackle it. The first is to offload the unsatisfied tasks of ECs to the remote cloud. Tong *et al*. [7] presented a hierarchical MEC structure among users, ECs and the cloud, where the tasks exceeding ECs' capacity will be offloaded to the cloud. Xiao *et al*. [8] studied the tradeoff between user QoE and fog nodes' power efficiency for three-layer fog computing system. However, this approach highly relies on a single EC, limiting the MEC performance. The second is to have edge cloud federation as a supplement to MEC. Assisted by virtualization technology, such practice which is also the main focus of this paper, can maintain flexible supply of distributed shared resources [10]. Chen *et al*. [11] leveraged computation offloading between ECs to minimize system-wide latency. The collaborative MEC platform was designed in [12] to incentivize ECs to form coalitions and share resources. However, most of prior work assumes that ECs can process whatever type of tasks regardless of service availability [3], [4], [6], which is central to the MEC system design subject to limited storage resources [10]. To break this barrier, we focus on a jointly optimized design of cooperative placement and scheduling for edge cloud federation.

Why Joint Cooperative Placement and Scheduling. There has been growing interest in designing service placement schemes for MEC [13]. Pasteris *et al*. [14] addressed the problem of placing multiple services to maximize the total reward. Wang *et al*. [15] studied the dynamic service placement problem for mobile microclouds with multiple users and service instances. Which services to place allows which type of tasks to be processed, thereby affecting scheduling performance. But none of the above solutions considered the interactions between service placement and task scheduling. Instead, we highlight the critical need for the joint cooperative placement and scheduling. There were similar works in the past that we can build up on. For example, Zeng *et al*. [16] systematically placed task images and scheduled tasks on ECs equipped with storage and computing resources. He *et al*. [10] optimized the joint placement and scheduling with the sharable (storage) and non-sharable (computing, transmission) resources. To reduce cost and improve stability, Farhadi *et al*. [17] proposed a two-time-scale MEC framework for joint placement and scheduling. Another focus of research has been to develop joint caching and scheduling schemes for MEC [20], [21]. By modeling collaboration across edge caches as content multicasts, Shukla *et al*. [22] considered a joint caching and request routing problem for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2021.3061602, IEEE Transactions on Mobile Computing

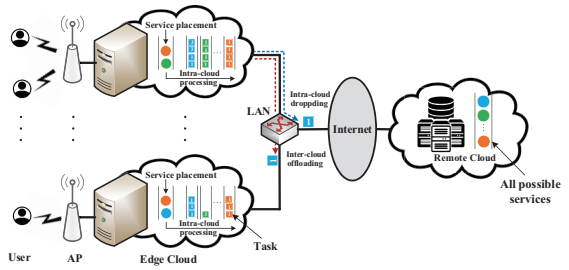


Fig. 1. Illustration of the edge cloud federation system. Different colors capture different types of services or tasks, and the numbers on squares denote user deadline demands.

cost minimization subject to cache capacity. It is worth noting that content caching and service placement may look similar as they both store a given resource and use it back when requested. They are, however, substantially different: the former focuses on storage constraints, while the latter has to take storage, computing and transmission constraints into account. All of the aforementioned studies typically either focus on spatial scheduling optimization, or assume a cooperative scenario where the ECs make decisions for global optimization. By contrast, our work pursues the spatially-temporally optimized cooperative control by capturing both user deadline preference and ECs' strategic behaviors, which produces new challenges for edge cloud federation system design.

Why Deadline-Driven Scheduling. There exists a large body of work for scheduling deadline-constrained tasks which need to be processed within the acceptable time interval [23], [24]. In practice, users usually show high heterogeneity in deadline tolerance, suggesting the potential performance boost if the distinction between time-critical and non-time-critical services is properly addressed. In cloud computing, Vamanan *et al*. [25] proposed a deadline-aware transport protocol which prioritizes near-deadline flows over far-deadline ones. Zheng *et al*. [26] studied the multi-resource allocation for deadline-sensitive tasks, validating the benefit of deadline awareness in scheduling. Our study highlights the deadline restriction in context of MEC. Li *et al*. [24] considered a joint assignment to maximize the ratio of offloaded tasks that meet the deadlines. To provide differentiated service provision, Katsalis *et al*. [18] proposed a SLA-driven scheduling method to minimize SLA violations for time-critical services. However, both of them focus on single-time-point optimization or finite differentiated service levels. In contrast, we target a combinatorial deadline-driven scheduling scheme, which is expected to: 1) leverage user deadline preference to optimize system efficiency across temporal domain; and 2) integrate edge cloud federation into strategic decision making for spatially-temporally optimized online control.

3 System Model and Problem Formulation

3.1 Edge Cloud Federation System

Consider a MEC provider with a set $\boldsymbol{N}$ of federated ECs densely deployed and connected by the same local area network (LAN). Each EC endowed with cloud-like computing and storage capabilities, is collocated with an access point (AP) and serves a dedicated set of users in its coverage[1]. The system runs in a time-slotted fashion for $t \in \boldsymbol{T} = \{0, \cdots, T-1\}$, where the slot is a much slower time scale than that of task scheduling [11].

1. Unlike cloud datacenters, ECs suffer from limited coverage due to physical limitations (e.g., limited transmit power), meaning that the user's task might not be finished within one EC if it moves out of the coverage [4], [10].

TABLE 2
Major Notations

Notation	Description
$\boldsymbol{N}$, $\boldsymbol{I}$, $\boldsymbol{R}$, $\boldsymbol{T}$	sets of ECs, users, services and slots
n, i, r, t	indexes of ECs, users, services and slots
r_i, n_i, t_i, d_i	requested service and EC, arrival time, deadline
λ^r, ϵ^r	expected data size, workload for per type-r task
S_n, f_n^r, $y^{\max}$	storage, computing and dropping capacity
E_n^r, $C[t]$	sharing budget, social cost
χ^r, η^r	income, dropping penalty for per type-r task
$c_{n,m}[t]$, $c_n^r[t]$	offloading and placement cost
$a_n^r[t]$, $\alpha_{n,i}[t]$	admission, offloading decision variables
$\beta_n^r[t]$, $x_{n,i}[t]$, $y_{n,i}[t]$	placement, processing, dropping decision variables
$\boldsymbol{D}_n^r[t]$, $\boldsymbol{A}_n^r[t]$	sets of service demands and admitted tasks
$\boldsymbol{Q}_n^r[t]$, $\boldsymbol{\alpha}_n^r[t]$	sets of queued tasks and offloaded tasks
Q_n^r, Z_n^r, H_n^r	task queues, virtual incentive and priority queues

The system is to provision R types of deadline-sensitive services. As an abstraction of applications, service is hosted by ECs and then requested by users in the form of tasks. Each EC hosts services of the same type in a time slot, which can change across different slots [21]. User $i \in \boldsymbol{I}$ randomly arrives at the system in slot $t_i \in \boldsymbol{T}$ by submitting one specific task of type $r_i \in \boldsymbol{R}$ via the EC $n_i \in \boldsymbol{N}$ covering (i.e., directly associated with) it. λ_i is the input data size of the task (in bits), and requires ϵ_i slots to process. Task processing doesn't need to be continuous [28]. The task can be processed in any slot as long as the total processing time meets ϵ_i before the deadline (i.e., maximum tolerable task completion delay), denoted by d_i with $0 < d_i \leq d^{\max}$, capturing users' heterogenous deadline tolerance towards service offerings. Formally, the task i submitted in slot $t \in \boldsymbol{T}$ can be specified by a *tuple* $\langle t_i, n_i, r_i, \lambda_i, \epsilon_i, d_i \rangle$. To simplify the system model, we assume that the tasks of the same type are similar in data size and workloads [27]. For any type-r task, the expected data size is λ^r (in bits), and the expected number of slots required to finish processing is ϵ^r, which is referred to the *workload*[2] of the task [27]. In particular, λ^r and ϵ^r are considered to be randomly drawn from $\mathcal{O}_\lambda \in [\lambda^{\min}, \lambda^{\max}]$ and $\mathcal{O}_\epsilon \in [\epsilon^{\min}, \epsilon^{\max}]$, respectively[3].

We target a joint cooperative mechanism with the main idea depicted in Fig. 1. Users can arrive in any slot by submitting heterogeneous tasks via associated APs. By leveraging the variations in workloads and resource costs among federated ECs, the MEC provider determines whether to admit newly arrived tasks, and if admitted, charges and offloads them to the ECs. For each EC, the provider judiciously decides which services to place under limited storage capacity. Given placement decisions, priority-based intra-cloud processing and dropping are performed with task priority captured by deadline urgency. Here "dropping" a task just *means not processing it on the network edge, and a dropped task will be processed in the remote cloud, typically at a higher cost*. For notation, the bold uppercase letters $\boldsymbol{X}$ denote sets, and non-bold letters x, X denote scalars, where the dependencies between relevant sets or scalars are captured by superscript $\hat{(\cdot)}$. Moreover, the superscript r and subscripts n, i denote service r, EC n and task i. To facilitate reading, we list the major notations in Table 2.

2. By modeling task workload like this, the duration of each slot for scheduling can be comparable to the time scale of service placement. Here we only focus on the single-time-scale representation, and a more general time scale representation will be discussed in Section 3.3.3.

3. This assumption is realistic especially for recurrent workloads. Take video streaming for example. Since user requests for this application always exhibit regular service demand patterns, it is possible to predict the input data size and processing time from the historical experience. Methods for quantifying task size and workload are beyond the scope of this paper.

3.2 Joint Cooperative Control Model

Similar to cloud datacenters [29], each EC consists of a front-end server and a back-end cluster. The former is responsible for inter-cloud scheduling including task admission and offloading, while the latter utilizes the provisioned resources to process the offloaded tasks, which can be cast into a joint strategy making for intra-cloud placement and scheduling.

3.2.1 Inter-Cloud Scheduling Control

1) Task admission. Users arrive at the system by submitting heterogenous task requests. Denote the set of type-r tasks that arrive at EC $n \in \boldsymbol{N}$ in slot $t \in \boldsymbol{T}$ as $\boldsymbol{D}_n^r[t]$ with the size $D_n^r[t] = |\boldsymbol{D}_n^r[t]| \leq D^{\max}$, capturing uneven service demands across ECs and slots. To prevent system overload, only a subset of tasks are admitted. We introduce the binary admission variable $a_n^r[t]$, where $a_n^r[t] = 1$ means EC n accepts all newly arrived type-r tasks in slot t, and $a_n^r[t] = 0$ otherwise. The set of type-r tasks admitted via n can be described as $\boldsymbol{A}_n^r[t]$, which is a subset of $\boldsymbol{D}_n^r[t]$.

2) Task offloading. Tasks will be offloaded to ECs once they are admitted. Denote $\boldsymbol{A}^r[t] = \cup_{n\in\boldsymbol{N}} \boldsymbol{A}_n^r[t]$ with the size $A^r[t] = |\boldsymbol{A}^r[t]|$ as the set of all type-r tasks admitted in slot t waiting to be offloaded. In this work, we focus on one-hop offloading case, where the tasks can only be offloaded once. Let $\alpha_{n,i}[t] \in \{0,1\}$ denote whether task i is offloaded to EC n in t. The set of type-r tasks offloaded to EC n in slot t is captured by $\boldsymbol{\alpha}_n^r[t] = \{i|\alpha_{n,i}[t] = 1, \forall i \in \boldsymbol{A}^r[t]\}$. The cost for offloading per bit workload out of EC m to EC n in slot t is $c_{n,m}[t]$, which is associated with the incurred energy consumption, and bandwidth usage and price. Here we consider a realistic environment that dynamically changes, where the transmission cost is uncertain and fluctuating since it depends on time-varying network conditions among ECs [6]. Since the offloading cost from the user to its requested EC is irrelevant to the joint cooperative control later, we will omit it for simplicity in the rest of this paper.

3.2.2 Intra-Cloud Placement and Scheduling Control

1) Service placement. ECs can only place a subset of services at the same time, under the limited capacity to store the service-related data/code. Placing a service in the EC allows this type of tasks to be processed. To satisfy volatile user demands, ECs have to judiciously decide which services to place. Define the binary variable $\beta_n^r[t] \in \{0,1\}$ to capture if service $r \in \boldsymbol{R}$ is placed by EC $n \in \boldsymbol{N}$ in slot $t \in \boldsymbol{T}$. The corresponding placement cost $c_n^r[t]$ (e.g., power consumption for running servers) is time-varying.

2) Task processing/dropping. Given user deadline tolerance, delays in service are acceptable if the tasks can be completed before deadlines. We model the potential dropping of a task when it fails to be fulfilled before the deadline. Among all provisioned services, the EC next determines which tasks to schedule. Denote $\boldsymbol{Q}_n^r[t]$ as the set of type-r tasks queued in EC n at the beginning of slot t. Since intra-cloud scheduling is performed during the slot when tasks are offloaded, all queued tasks including newly offloaded ones might be processed or dropped in this slot. For convenience, denote $\hat{\boldsymbol{Q}}_n^r[t] = \boldsymbol{Q}_n^r[t] \vee \boldsymbol{\alpha}_n^r[t]$ with the size $\hat{Q}_n^r[t] = |\hat{\boldsymbol{Q}}_n^r[t]|$, capturing the set of type-r tasks that can be scheduled by EC n in t. Both updates of $\boldsymbol{Q}_n^r[t]$ (at the end of $t-1$) and $\boldsymbol{\alpha}_n^r[t]$ (at the beginning of t) may bring about the dynamics of $\hat{\boldsymbol{Q}}_n^r[t]$.

Let $x_{n,i}[t] \in \{0,1\}$ and $y_{n,i}[t] \in \{0,1\}$ be the indicators about whether task $i \in \hat{\boldsymbol{Q}}_n^r[t]$ is processed and dropped in slot t. The total number of slots of task i that are served by slot t is $x_i[t] = \sum_{\tau=1}^{t} x_{n,i}[\tau]$. We obtain the processing rate for type-r tasks, denoted by $x_n^r[t] = \sum_{i\in\hat{\boldsymbol{Q}}_n^r[t]} x_{n,i}[t]$. Similarly, the dropping rate in terms of service r acquired by n in slot t is $y_n^r[t] = \sum_{i\in\hat{\boldsymbol{Q}}_n^r[t]} (\epsilon^r - x_i[t]) y_{n,i}[t]$. During slot t, the task i will be deleted from $\hat{\boldsymbol{Q}}_n^r[t]$ if it is completely processed or dropped, i.e., $\hat{\boldsymbol{Q}}_n^r[t] \backslash \{i\}$ if $x_i[t] = \epsilon^r$ or $y_{n,i}[t] = 1$. At the end of slot t, $\boldsymbol{Q}_n^r[t]$ is updated to the set of remaining tasks in $\hat{\boldsymbol{Q}}_n^r[t]$.

3.3 Problem Formulation

3.3.1 Constraints of The Problem

1) *Capacity Constraint.* We highlight the limited storage and scheduling capacities of ECs as follows:

$$\sum_{r\in\boldsymbol{R}} \beta_n^r[t] \leq S_n, \ \forall n \in \boldsymbol{N}, t \in \boldsymbol{T}, \tag{1}$$

$$x_n^r[t] \leq \beta_n^r[t] f_n^r, \ \forall r \in \boldsymbol{R}, n \in \boldsymbol{N}, t \in \boldsymbol{T}, \tag{2}$$

$$y_n^r[t] \leq \ y^{\max}, \quad \forall r \in \boldsymbol{R}, n \in \boldsymbol{N}, t \in \boldsymbol{T}, \tag{3}$$

where (1) ensures the number of services that the EC can place in one slot is no more than its *storage capacity*; (2) indicates any type-r task can be processed only if the service r is placed and at most f_n^r (referred to the *maximum processing rate*) tasks are processed in one slot for the guaranteed best-effort service; (3) specifies the *maximum dropping rate* of EC n for type-r tasks by placing an upper bound $y^{\max}$ for the amount of dropped workloads.

2) *Deadline Constraint.* Given any task set $\hat{\boldsymbol{Q}}_n^r[t]$, every task i only gets either processed or dropped before its deadline d_i, i.e.,

$$t \cdot x_{n,i}[t] \leq t_i + d_i, \forall i \in \hat{\boldsymbol{Q}}_n^r[t], r \in \boldsymbol{R}, n \in \boldsymbol{N}, t \in \boldsymbol{T}, \tag{4}$$

$$t \cdot y_{n,i}[t] \leq t_i + d_i, \forall i \in \hat{\boldsymbol{Q}}_n^r[t], r \in \boldsymbol{R}, n \in \boldsymbol{N}, t \in \boldsymbol{T}. \tag{5}$$

3) *Workload Constraint.* For any queued type-r task i, there is no extra value for EC n to serve it more than ϵ^r slots, i.e.,

$$x_i[t] \leq \epsilon^r, \forall i \in \hat{\boldsymbol{Q}}_n^r[t], r \in \boldsymbol{R}, n \in \boldsymbol{N}, t \in \boldsymbol{T}. \tag{6}$$

4) *Conservation Constraint.* This constraint enforces that in each slot, the total workload offloaded by the EC (including its retained tasks) equals its admitted workload, i.e.,

$$\sum_{i\in\boldsymbol{A}_n^r[t]} \sum_{m\in\boldsymbol{N}} \alpha_{m,i}[t] = a_n^r[t] D_n^r[t], \forall r \in \boldsymbol{R}, n \in \boldsymbol{N}, t \in \boldsymbol{T}. \tag{7}$$

The intuition is that each admitted task is guaranteed to be offloaded right away, which is vital for deadline-sensitive services.

5) *Stability Constraint.* The number of type-r tasks offloaded to n in each slot cannot exceed its maximum processing rate, i.e.,

$$\sum_{i\in\boldsymbol{A}^r[t]} \alpha_{n,i}[t] \leq f_n^r, \forall r \in \boldsymbol{R}, n \in \boldsymbol{N}, t \in \boldsymbol{T}. \tag{8}$$

6) *Incentive Constraint.* This constraint is specified for resource consumption in sharing offloading. Denote $\hat{\boldsymbol{A}}_n^r[t] = \cup_{m\in\boldsymbol{N}\backslash\{n\}} \boldsymbol{A}_m^r[t]$ as the set of type-r tasks admitted by peers that EC n can cooperate in slot t. To explore collaborative computing potentials, MEC provider always forces ECs with sufficient resources to help peers with task offloading frequently. Such sharing, however, depletes remaining resources for its own use sooner than expected and incurs additional cost (for receiving requests and returning responses), thereby reducing ECs' sharing willingness. To avoid this, we introduce the sharing budget E_n^r for EC n on its helping peers offload type-r tasks over the time horizon $\boldsymbol{T}$, i.e.,

$$\frac{1}{T} \sum_{t\in\boldsymbol{T}} e_n^r[t] \leq E_n^r, \forall r \in \boldsymbol{R}, n \in \boldsymbol{N}, \tag{9}$$

where $e_n^r[t] = \sum_{i\in\hat{\boldsymbol{A}}_n^r[t]} \alpha_{n,i}[t] \leq (N-1) D^{\max}$ captures n's total resource consumption for helping peers with type-r tasks in t.

Remark. The joint cooperative control decisions are not independent. This can be reflected in the following aspects: (i) under capacity, deadline, workload constraints, intra-cloud control interacts with each other; (ii) conservation constraint shows inter-cloud offloading is performed subject to admission control; (iii) deadline and workload constraints indicate intra-cloud scheduling is conducted on the set $\hat{\boldsymbol{Q}}_n^r[t]$, i.e., the newly offloaded tasks can also be processed or dropped in current slot. (iv) (2) suggests task processing is performed on the basis of placement control.

3.3.2 Online Joint Cooperative Control Problem

In the edge federation system, the social welfare generated in each slot is the difference between total service income and total service cost. The total service income of admitting the tasks of different types in slot t is $S^I[t] = \sum_{n,r} \chi^r D_n^r[t] a_n^r[t]$, where $\chi^r \in (0, \chi^{\max}]$ is the income for admitting per type-r task. The instantaneous total service cost is defined as the sum of placement cost that covers the resource consumption for processing tasks, and scheduling cost consisting of inter-cloud offloading cost and expenditure on penalty (e.g, resource consumption for offloading to the remote cloud and processing there) occurred if there are dropped tasks. That is, $S^C[t] = \sum_{n,r} \left(c_n^r[t]\beta_n^r[t] + \sum_{i \in \hat{\boldsymbol{A}}_n^r[t]} \lambda^r c_{n,n_i}[t]\alpha_{n,i}[t] + \sum_{i \in \hat{\boldsymbol{Q}}_n^r[t]} \eta^r y_{n,i}[t] \right)$, where η^r is the penalty for dropping per type-r task.

Maximizing the social welfare over the time horizon $\boldsymbol{T}$ is equivalent to minimizing the overall (average) social cost over all slots, i.e., $\frac{1}{T}\sum_t C[t] = \frac{1}{T}\sum_t (S^C[t] - S^I[t])$. The goal of the MEC provider is to make joint cooperative control to minimize the overall social cost while respecting the aforementioned constraints. The problem of interest can be formulated as follows:

$$\begin{aligned} \min \quad & \frac{1}{T}\sum_{t \in \boldsymbol{T}} C[t] \\ \text{s.t.} \quad & \text{Constraints (1)-(9).} \end{aligned} \tag{10}$$

Actually, (10) is approximately equivalent to maximizing the number of users served on the edge, where the dropped tasks are offloaded to the remote cloud for processing at a higher cost.

Edge cloud federation enables tasks to be offloaded away from overloaded or high-cost ECs to lightly-loaded or low-cost ones, offering opportunities for cost reduction or task completion. In practice, users usually show heterogenous deadline tolerance, which begs the question: how to choose between more valuable tasks with large deadlines and less valuable yet more emergent tasks subject to limited resources. Deadline-driven scheduling is promising to best satisfy user demands [26], whose main idea is to schedule a few tasks with small deadlines besides those tasks with high values in each slot. If edge cloud federation acts as spatial optimization strategy in a single slot, then surely deadline-driven scheduling allows workloads to be migrated across different times, thus optimizing the system efficiency across temporal domain. Despite these advantages of deadline awareness, we notice that it is still necessary to study deadline-oblivious algorithms since they are often easier to implement and have wider applications, especially when no exact or complete task deadline information is available. Here deadline-oblivious suggests the deadline information is not used in decision making. This case actually serves as the spatially optimized benchmark. In what follows, we propose JCPS, a novel online joint cooperative placement and scheduling framework for two cases of edge cloud federation.

3.3.3 Discussions

Time scale representation. This work is to offer a deadline-driven design for cost-efficient MEC systems via joint cooperative placement and scheduling, whose time scales are comparable with task workloads in terms of processing time [28]. Notice that the proposed solution cannot be directly applied to general computation tasks, whose workloads are captured by the required CPU cycles. In this case, task processing takes shorter time than placing services, and adjusting placement as frequently as scheduling would incur high operation cost and system instability. A promising alternative is two-time-scale representation, where the time scales of placement is split from scheduling. There have been studies on two-time-scale optimization designs [29]. Farhadi *et al*. [17] proposed a two-time-scale framework for joint placement and scheduling. But due to correlations across decision intervals, they only focus on the joint control in one frame, and the formulation of multi-frame optimization problem is simply presented without further analysis. As a future work, it is worth studying the temporal decision correlations in two-time-scale system designs.

User mobility. Our proposed solutions can also be extended for the case where users move erratically during task processing. The assumption is that users only move within the coverage of requesting ECs. The joint cooperative control is independent from user mobility due to the unchanged correspondence between users and requesting ECs. But if users move out the coverage, that's another story. Imagine after submitting a request to EC n_i, user i moves to the coverage of another EC n. If the requested service is still placed in n_i, the perceived latency would greatly increase due to the increased transmission distance. To optimize user experience, the service profile should be actively migrated across multiple ECs to follow user mobility, yet which would incur a huge additional cost. How to navigate such performance-cost tradeoff has been studied in earlier works [2]. A full analysis of dynamic service migration under user mobility is out of the scope of this work. But it surely will be interesting for future research to study the mobility-aware joint cooperative control problem.

4 Online Joint Cooperative Placement and Scheduling for Deadline-Oblivious Case

We begin with a simple but classical setting, the deadline-oblivious case [10], [16], and develop a joint cooperative placement and scheduling mechanism (DO-JCPS), using subroutine $\mathcal{A}_{slot}$ running in each slot. For guaranteed service quality, the following locality constraint is typically highlighted in related works [10] by enforcing tasks can only be offloaded to ECs that place the desired services, i.e.,

$$\alpha_{n,i}[t] \leq \beta_n^r[t], \forall i \in \boldsymbol{A}^r[t], r \in \boldsymbol{R}, n \in \boldsymbol{N}, t \in \boldsymbol{T}. \tag{11}$$

4.1 Design of Approximation Algorithm $\mathcal{A}_{slot}$

We first formulate the single-slot deadline-oblivious joint cooperative placement and scheduling problem (sDO-JCPS):

$$\begin{aligned} \min \quad & \sum_{n,r} \left(c_n^r[t]\beta_n^r[t] + \sum_{i \in \hat{\boldsymbol{A}}^r[t]} \hat{c}_{n,i}[t]\alpha_{n,i}[t] \right) \\ \text{s.t.} \quad & \text{Constraints (1)-(8), (11),} \\ & a_n^r[t], \alpha_{n,i}[t], \beta_n^r[t] \in \{0,1\}, \forall i \in \boldsymbol{A}^r[t], r \in \boldsymbol{R}, n \in \boldsymbol{N}, \end{aligned} \tag{12}$$

which warrants some discussions, especially for those different from (10). Conservation constraints allow the elimination of

$D_n^r[t]$ from the objective, and scheduling cost $\lambda^r c_{n,n_i}[t] - \chi^r$ is replaced by regulated cost $\hat{c}_{n,i}[t]$ (to be specified later). To ensure fairness, tasks are processed in order and dropped if exceeding the deadlines. Thus, processing and dropping decisions are the consequence of placement ones, making intra-cloud joint control to be a pure placement problem. Incentive constraints are removed since this long-term constraint couples shared offloading (short for "sharing resources for helping other ECs with task offloading") decisions across all slots, beyond the scope of single-slot problem.

For any specific type of service, offloading tasks to ECs falls short into the capacitated facility location problem (CFLP), which has proven $\mathcal{NP}$-hard [30]. Specifically, in CFLP, there is a set $\boldsymbol{F}$ of potential facilities and a set $\boldsymbol{U}$ of users. Each facility $i \in \boldsymbol{F}$ has a capacity s_i. Establishing facility i incurs opening cost f_i and serving user $u \in \boldsymbol{U}$ incurs service cost $c_{i,u}$. The objective is opening a subset of facilities to serve users with minimal cost under capacity constraints. Actually, placing service r among ECs can be viewed as a facility location problem, where opening a facility parallels placing one service in an EC and offloading tasks to it. Hence, sDO-JCPS for any type of service is $\mathcal{NP}$-hard. A problem of APX-hardness is a set of $\mathcal{NP}$-hard problems that can be solved via approximation methods [31]. Given APX-hardness of sDO-JCPS, we resort to approximation algorithms to obtain a near-optimal solution with computation efficiency.

Definition 2. *An algorithm is called a (γ_p, γ_s)-approximation algorithm for sDO-JCPS in (12), if for every instance $\mathcal{I}$ of sDO-JCPS and every solution $\boldsymbol{\phi}$ of $\mathcal{I}$ with placement cost $C_{place}(\boldsymbol{\phi})$ and scheduling cost $C_{schedule}(\boldsymbol{\phi})$, the cost of the solution found by the algorithm is at most $\gamma_p C_{place}(\boldsymbol{\phi}) + \gamma_s C_{schedule}(\boldsymbol{\phi})$.*

Combining stability and locality constraints yields $\sum_{i \in \boldsymbol{A}^r[t]} \alpha_{n,i}[t] \le \beta_n^r[t] f_n^r$. Given constant $\sigma \in [0,1]$, we deduce the underlying condition, i.e., $\sum_{n,r,i} \frac{\sigma c_n^r[t]}{f_n^r} \alpha_{n,i}[t] \le \sum_{n,r} \sigma c_n^r[t] \beta_n^r[t]$. On this basis, we generate the following relaxation problem:

$$\min \sum_{n,r} \left((1-\sigma) c_n^r[t] \beta_n^r[t] + \sum_{i \in \hat{\boldsymbol{A}}^r[t]} \alpha_{n,i}[t] \left(\frac{\sigma c_n^r[t]}{f_n^r} + \hat{c}_{n,i}[t] \right) \right) \tag{13}$$

$$\text{s.t.} \quad \text{Constraints (1)-(7), (11)},$$
$$a_n^r[t], \alpha_{n,i}[t], \beta_n^r[t] \in \{0,1\}, \forall i \in \boldsymbol{A}^r[t], r \in \boldsymbol{R}, n, \in \boldsymbol{N},$$

which establishes a lower bound of sDO-JCPS. By adopting similar techniques used in [32], we obtain the following lemma.

Lemma 1. *Let $\sigma = \frac{\gamma_s}{\gamma_p + \gamma_s} \in [0,1]$. Any (γ_p, γ_s)-approximation algorithm that solves (13) also yields a $(\gamma_p + \gamma_s)$-approximation algorithm for sDO-JCPS.*

Proof. See Appendix A. □

Lemma 1 provides the convenience to consider only problem (13) hereafter, and its integer linear formulation allows us to apply branch-and-bound algorithm [5], [33] to solve it optimally. The solution called $\mathcal{A}_{slot}$ is shown in Alg. 1. Denote U^* as the optimal value of sDO-JCPS problem's objective function, and $\boldsymbol{P}$ as the problem set with a lower bound L and an upper bound U of U^*. It is initialized as p_0 with integer constraints relaxed to linear ones, i.e., $a_n^r[t], \alpha_{n,i}[t], \beta_n^r[t] \in [0,1]$. For any $p \in \boldsymbol{P}$, the solution by linear program relaxation serves as a lower bound L_p of the solution to p_0. The branch-and-bound process is divided into multiple iterations. In each iteration, we locate problem p with the minimum L_p and set $L = L_p$. Any feasible solution to p serves as an upper bound of U_p and the tightest upper bound of U can be updated (lines 4-17). If L is no less than U, U^* is returned. Otherwise, decompose p with two subproblems p_0, p_1 by branching binary variables (lines 10-13). Different from traditional branch-and-bound algorithms, sDO-JCPS has some problem-specific characteristics: by fixing one variable, other variables could be fixed right away as many as possible due to their correlation constraints. For example, when $a_m^r[t] = 0$, $\alpha_{n,i}[t] = 0, \forall i \in \boldsymbol{A}_m^r[t]$ since no task is admitted. This feature allows us to design an efficient acceleration approach to reduce complexity of (13)'s relaxed problem. We fix variables in order of $a_n^r[t]$, $\beta_n^r[t]$, $\alpha_{n,i}[t]$, and for each type of variables, they are fixed in decreasing order of η_p, the result of p's relaxed problem. By adopting a greedy approach like rounding, the number of iterations to achieve the desired solution is reduced.

Algorithm 1: $\mathcal{A}_{slot}$ for sDO-JCPS Problem

1 Set L_p as the solution to (13), $\boldsymbol{P} = \{p_0\}, U = \infty$;
2 **while** $\boldsymbol{P} \neq \emptyset$ **do**
3 Select $p \in \boldsymbol{P}$ with minimal L_p and result η_p; Let $L = L_p$; Set U_p as the solution to p by rounding;
4 **if** $U_p < U$ **then**
5 $U^* = U_p$, $U = U_p$;
6 **if** $L \ge U$ **then**
7 Return the optimal solution U^*;
8 **else**
9 Remove all problems $p' \in \boldsymbol{P}$ with $L_{p'} \ge U$;
10 **foreach** $x \in \{a_n^r[t], \alpha_{n,i}[t], \beta_n^r[t]\}$ **do**
11 **if** *there are unfixed x in p* **then**
12 Set x with maximal η_p by y;
13 Set $y = \lceil y \rceil$ in p_1, $y = \lfloor y \rfloor$ in p_0; fix associated variables by constraints (1)-(7), (11);
14 **foreach** $i \in \{0,1\}$ **do**
15 Solve p_i by relaxing unfixed variables and obtain L_{p_i};
16 **if** $L_{p_i} < U$ **then**
17 Put p_i into $\boldsymbol{P}$.

Theorem 1. *$\mathcal{A}_{slot}$ is a 2-approximation for sDO-JCPS.*

Proof. See Appendix B. □

4.2 Design of Deadline-Oblivious Mechanism DO-JCPS

We next develop an online mechanism DO-JCPS to cope with the multi-slot problem (10) without deadline awareness. The algorithm works as follows (see Alg. 2). In each slot t, only ECs' workload and resource cost at and before t are known, i.e., the joint cooperative control is conducted without future knowledge. Given the placement decisions, the queued tasks are processed in order until exceeding the deadlines or satisfying the workloads (lines 9-16). Under incentive constraint, ECs have a sharing budget for helping others with task offloading over $\boldsymbol{T}$, making shared offloading decisions coupled over time. If providing shared offloading service in one slot, the EC's residual shared capacity will decrease and it will be less likely to be selected for offloading future tasks. A promising approach is to scatter the shared offloading capacity intelligently over $\boldsymbol{T}$ [34]. We introduce regulation factor $\mu_n^r[t]$ to adjust n's scheduling cost for helping offload type-r tasks in slot t as the input to $\mathcal{A}_{slot}$. Since n's shared capacity initialized to TE_n^r decreases over time, we initialize $\mu_n^r[t]$ as 0 (line 1) and scale it up for any selected EC.By regulating scheduling cost (line

Algorithm 2: Online Mechanism DO-JCPS

1 $\mu_n^r[0]=0, \beta_n^r[0]=0, x_{i,0}=0, \hat{\boldsymbol{Q}}_n^r[0]=\emptyset, \forall i,r,n$;
2 $\delta=\max_{r,n,t}\frac{TE_n^r}{e_n^r[t]+1}$;
3 **for** *Each slot* $t\in\boldsymbol{T}$ **do**
4 $\hat{c}_{n,i}[t]=\lambda^r c_{n,n_i}[t]-\chi^r+\beta_n^r[t-1]\mu_n^r[t-1], \forall i,r,n$;
5 Run $\mathcal{A}_{slot}$ to obtain decisions $a_n^r[t], \alpha_{n,i}[t], \beta_n^r[t]$;
6 **for** *Each cloud* $n\in\boldsymbol{N}$*, service* $r\in\boldsymbol{R}$ **do**
7 $\hat{\boldsymbol{Q}}_n^r[t]\leftarrow\hat{\boldsymbol{Q}}_n^r[t-1]\vee\boldsymbol{\alpha}_n^r[t]$;
8 **if** $\beta_n^r[t]=1$ **then**
9 Select the first f_n^r tasks from $\hat{\boldsymbol{Q}}_n^r[t]$ by t_i;
10 **for** *Each task* $i\in\hat{\boldsymbol{Q}}_n^r[t]$ **do**
11 **if** i *is selected* **then**
12 $x_i[t]=x_i[t-1]+1$;
13 **else**
14 $x_i[t]=x_i[t-1]$;
15 $\hat{\boldsymbol{Q}}_n^r[t]\leftarrow\hat{\boldsymbol{Q}}_n^r[t]\backslash\{i\}$ if $x_i[t]\geq\epsilon^r$ or $t-t_i\geq d_i$;
16 $e_n^r[t]=\sum_{i\in\hat{\boldsymbol{A}}_n^r[t]}\alpha_{n,i}[t], \forall r,n$;
17 **if** $e_n^r[t]>0$ **then**
18 $\mu_n^r[t]=\mu_n^r[t-1](1+\frac{\beta_n^r[t-1]}{\gamma TE_n^r})+\frac{\min_m \lambda^r c_{n,m}[t]-\chi^r}{\gamma\delta TE_n^r}$;
19 **else**
20 $\mu_n^r[t]=\mu_n^r[t-1]$.

4), $\mu_n^r[t]$ can be interpreted as unit cost for using residual capacity. How to scale it up concerns whether to respect residual shared capacity properly. If setting it to a large value, $\mathcal{A}_{slot}$ will be too conservative in selecting ECs with low residual capacity in early slots and some cost-effective ECs may end up with being unallocated, which causes the waste of resources; otherwise, the cheap ECs' capacity will be used up too early, forcing tasks to be offload to expensive ones later on. The updating rule of $\mu_n^r[t]$ is shown in lines 17-20, where other ECs' costs remain intact.

In Alg. 2, the outer for loop runs T slots. In each slot, calculate the regulated cost in $\mathcal{O}(A^r[t])$ steps and apply the branch-and-bound algorithm to determine the joint cooperative control with complexity of $\mathcal{O}(2^N)$ [5]. After that, the inner for loop is terminated after NR iterations, in each of which select tasks to process in $\mathcal{O}(\hat{Q}_n^r[t]\log(\hat{Q}_n^r[t]))$ steps and perform updating in $\mathcal{O}(\hat{Q}_n^r[t]+A^r[t])$ steps. Finally, update the regulation factor in $\mathcal{O}(N)$ steps. Overall, the complexity of Alg. 2 is $\mathcal{O}(T(A^r[t]+2^N+NR(\hat{Q}_n^r[t](1+\log(\hat{Q}_n^r[t]))+A^r[t]+N)))$.

Lemma 2. *Alg. 2 yields a feasible solution to problem (*10*).*

Proof. See Appendix C. □

Theorem 2. *The competitive ratio of Alg. 2 is* $\frac{2\delta}{\delta-1}$.

Proof. See Appendix D. □

The competitive ratio of Alg. 2 depends on δ, the maximum ratio between ECs' shared capacity and workload in one slot. When $\delta\to\infty$, the competitive ratio approaches 2, indicating if the shared workload in any slot is much smaller than shared capacity, the cost increase induced by online loss is negligible.

5 Online Joint Cooperative Placement and Scheduling for Deadline-Driven Case

In this section, we study the benefit of deadline awareness in joint cooperative control. To proceed, we first reformulate the deadline constraints (4)-(5) such that the heterogeneity of user deadline tolerance can be fully exploited. The main idea is to prioritize the popular or time-critical services and tasks in intra-cloud online control. Given any task set $\hat{\boldsymbol{Q}}_n^r[t]$, the scheduling ranking $\boldsymbol{R}_n^r[t]=\left\{R_n^r(1),\cdots,R_n^r(\hat{Q}_n^r[t])\right\}$ for EC n is a permutation of $\{1,\cdots,\hat{Q}_n^r[t]\}$. The priority constraint holds for type-r tasks with diverse deadline demands if $\boldsymbol{R}_n^r[t]$ satisfies

$$R_n^r(i)\begin{cases}<R_n^r(j), & \text{if } d_i<d_j,\\ \geq R_n^r(j), & \text{if } d_i\geq d_j,\end{cases} \tag{14}$$

for $i\in\{1,\cdots,\hat{Q}_n^r[t]-1\}$, $j\in\{i,\cdots,\hat{Q}_n^r[t]\}$. $R_n^r(i)=R_n^r(j)$ if two tasks share the same deadline demand and are submitted by one user. The service priority imposed by workload and deadline urgency drives time-critical placement and scheduling.

5.1 Deadline-Driven Joint Cooperative Placement and Scheduling Mechanism for Social Cost Minimization

Using Lyapunov techniques [35], we propose a deadline-driven joint cooperative solution by converting (10) with spatially-temporally coupled decisions to one-shot minimization problem. We're interested in the scenario where the online control among ECs is coordinated in a centralized way (this subsection) and the scenario where ECs make strategic decisions (next subsection).

5.1.1 Queue Dynamics

1) Task Queue: Each EC maintains R task queues, each of which corresponds to one specific type of service. Let the queue backlog $Q_n^r[t]$ denote the amount of type-r task workloads queued in EC n's back-end cluster at the beginning of slot t. Intuitively, stochastic inter-cloud and intra-cloud controls may bring about the dynamics of task queue backlog, which can be written as

$$Q_n^r[t+1]=\max\left\{Q_n^r[t]-x_n^r[t]-y_n^r[t]+\epsilon^r\sum_{i\in\boldsymbol{A}^r[t]}\alpha_{n,i}[t],0\right\}. \tag{15}$$

A queue is stable only if it has a bounded time-averaged backlog [35], i.e., $\bar{Q}_n^r=\limsup_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{Q_n^r[t]\}<\infty$.

2) Virtual Incentive Queue: For incentive constraints, we introduce a virtual incentive queue Z_n^r for EC n associated with service r. The backlog initialized to $Z_n^r[0]=0$ is updated as

$$Z_n^r[t+1]=\max\left\{Z_n^r[t]+e_n^r[t]-E_n^r,0\right\}. \tag{16}$$

3) Virtual Priority Queue: Under constraint (14), tasks with urgent deadlines are prioritized in joint cooperative control, realizing cost reduction and load balancing in temporal domain. Inspired by [36], we maintain a dynamic priority weight for each queued task i as $\omega_i[t]=\sigma e^{t-t_i-d_i}, \forall t_i\leq t\leq t_i+d_i$. The more urgent i's deadline is, the higher priority weight it will get. The priority weight is initially $\omega_i[t_i]=\sigma e^{-d_i}$ once being admitted, and reaches a maximum $\omega_i[t_i+d_i]=\sigma$ when it expires in slot t_i+d_i. The time evolution of priority weight can be captured as

$$\omega_i[t+1]=\omega_i[t]\cdot e=\omega_i[t]+\omega_i[t]\cdot(e-1). \tag{17}$$

We introduce a virtual priority queue H_n^r with backlog $H_n^r[t]=\sum_{i\in\boldsymbol{Q}_n^r[t]}\omega_i[t]$, capturing the overall urgency of queued type-r tasks. In particular, the backlog initialized to $H_n^r[0]=0$ evolves as

$$H_n^r[t+1]=\max\left\{H_n^r[t]+\xi_n^r[t]-\hat{x}_n^r[t]-\hat{y}_n^r[t]+\sum_{i\in\boldsymbol{A}^r[t]}\hat{\alpha}_{n,i}[t],0\right\}, \tag{18}$$

where $\xi_n^r[t] = \sum_{i\in\boldsymbol{Q}_n^r[t]}(e-1)\omega_i[t]$, $\hat{x}_n^r[t] = \sum_{i\in\boldsymbol{x}_n^r[t]} e\omega_i[t] = \sum_{i\in\hat{\boldsymbol{Q}}_n^r[t]} e\omega_i[t]x_{n,i}[t]$, $\hat{y}_n^r[t] = \sum_{i\in\boldsymbol{y}_n^r[t]} e\omega_i[t] = \sum_{i\in\hat{\boldsymbol{Q}}_n^r[t]} e(\epsilon^r - x_i[t])\cdot\omega_i[t]y_{n,i}[t]$, $\hat{\alpha}_{n,i}[t] = e\epsilon^r\omega_i[t]\alpha_{n,i}[t]$. Here $\boldsymbol{x}_n^r[t]$, $\boldsymbol{y}_n^r[t]$ are the sets of processed and dropped tasks associated with $x_n^r[t]$, $y_n^r[t]$.

5.1.2 Lyapunov Optimization

We define the perturbed Lyapunov function in terms of $\boldsymbol{\Theta}[t] = [\boldsymbol{Q}[t], \boldsymbol{Z}[t], \boldsymbol{H}[t]]$ as $L(\boldsymbol{\Theta}[t]) = \frac{1}{2}\|\boldsymbol{Q}[t]-\boldsymbol{\theta}\| + \frac{1}{2}\|\boldsymbol{Z}[t]\| + \frac{1}{2}\|\boldsymbol{H}[t]\|$, where $\boldsymbol{\theta} = \theta_n^r \cdot \mathbf{1}^{N\times R}$ with θ_n^r being perturbation parameters. To keep system stable, we introduce one-shot conditional Lyapunov drift as $\Delta(\boldsymbol{\Theta}[t]) = \mathbb{E}\{L(\boldsymbol{\Theta}[t+1]) - L(\boldsymbol{\Theta}[t])|\boldsymbol{\Theta}[t]\}$, capturing expected changes in quadratic function of backlogs over each slot. We incorporate social cost into Lyapunov drift, providing network stability and cost minimization jointly. The objective of D-JCPS is to minimize an upper bound of the drift-plus-cost function, i.e.,

$$\min \quad \Delta(\boldsymbol{\Theta}[t]) + V\mathbb{E}\{C[t]|\boldsymbol{\Theta}[t]\}, \tag{19}$$

where importance weight $V > 0$ is used to capture how much we emphasize on cost minimization compared to system stability.

Lemma 3. *Denote $\widetilde{Q}_n^r[t] = Q_n^r[t] - \theta_n^r$ and $\widetilde{H}_n^r[t] = e^2 H_n^r[t]$. Under any feasible online joint cooperative control, we have*

$$\begin{aligned}
&\Delta(\boldsymbol{\Theta}[t]) + V\mathbb{E}\{C[t]|\boldsymbol{\Theta}[t]\} \\
&\le B_1 + B_2[t] - \sum_{n,r}\mathbb{E}\left\{V\chi^r D_n^r[t]a_n^r[t]|\boldsymbol{\Theta}[t]\right\} \\
&+ \sum_{n,r}\mathbb{E}\left\{\sum_{i\in\boldsymbol{A}^r[t]}\alpha_{n,i}[t](\widetilde{Q}_n^r[t] + \widetilde{H}_n^r[t]\omega_i[t])|\boldsymbol{\Theta}[t]\right\} \\
&+ \sum_{n,r}\mathbb{E}\left\{\sum_{i\in\hat{\boldsymbol{A}}_n^r[t]}\alpha_{n,i}[t](Z_n^r[t] + V\lambda^r c_{n,n_i}[t])|\boldsymbol{\Theta}[t]\right\} \\
&- \sum_{n,r}\mathbb{E}\left\{\sum_{i\in\hat{\boldsymbol{Q}}_n^r[t]}x_{n,i}[t](\widetilde{Q}_n^r[t] + \widetilde{H}_n^r[t]\omega_i[t]) - Vc_n^r[t]\beta_n^r[t]|\boldsymbol{\Theta}[t]\right\} \\
&- \sum_{n,r}\mathbb{E}\left\{\sum_{i\in\hat{\boldsymbol{Q}}_n^r[t]}y_{n,i}[t](\widetilde{Q}_n^r[t] + \widetilde{H}_n^r[t]\omega_i[t] - V\eta^r)|\boldsymbol{\Theta}[t]\right\},
\end{aligned} \tag{20}$$

where $B_1 = \frac{1}{2}\sum_{n,r}[(1+e^2\sigma^2)(f_n^r + y^{\max})^2 + (1+e^2\sigma^2)(f_n^r)^2 + (E_n^r)^2 + (N-1)^2(D_n^{r,\max})^2 + \sigma^2(e^2-1)(Q_n^{r,\max})^2]$ is a positive finite constant, and $B_2[t] = -\sum_{n,r} Z_n^r[t]E_n^r$ is a known constant in slot t since the incentive queue backlog values are known at t.

Proof. See Appendix E. □

By Lemma 3, minimizing the drift-plus-cost in (19) is equivalent to minimizing the right-hand-side (RHS) of (20). The idea of D-JCPS is to minimize it subject to constraints (1)-(14), achieving the tradeoff between queue stability and cost minimization.

5.1.3 Centralized Joint Cooperative Control Policy

A careful investigation of the RHS of (20) reveals that (10) can be equivalently decoupled into three independent optimization.

1) Inter-Cloud Scheduling: The inter-cloud scheduling control for different types of services is independent. For each service $r \in \boldsymbol{R}$, decisions on $a_n^r[t]$ and $\alpha_{n,i}[t]$ can be made by solving

$$\begin{aligned}
\max \quad & \sum_n\left(\chi^r D_n^r[t]a_n^r[t] - \sum_{i\in\boldsymbol{A}^r[t]}W_{n,i}[t]\alpha_{n,i}[t]\right) \\
\text{s.t.} \quad & \sum_{i\in\boldsymbol{A}^r[t]}\alpha_{n,i}[t] \le f_n^r, \forall n\in\boldsymbol{N}, \\
& \sum_{i\in\boldsymbol{A}_n^r[t]}\sum_{m\in\boldsymbol{N}}\alpha_{m,i}[t] = D_n^r[t]a_n^r[t], \forall n\in\boldsymbol{N}, \\
& a_n^r[t], \alpha_{n,i}[t]\in\{0,1\}, \forall i\in\boldsymbol{A}^r[t], n\in\boldsymbol{N},
\end{aligned} \tag{21}$$

where $W_{n,i}[t] = \frac{1}{V}(\widetilde{Q}_n^r[t] + \widetilde{H}_n^r[t]\omega_i[t] + Z_n^r[t] + V\lambda^r c_{n,n_i}[t])$ denotes the *equivalent offloading cost* for offloading task i to EC n in slot t. This problem seems a little complicated due to the coupling of $a_n^r[t]$ and $\alpha_{n,i}[t]$. A feasible approach is backward induction, i.e., to convert it to a pure offloading problem and on that basis, to address the admission problem. Given admission decisions, (21) can be rewritten as the pure offloading problem:

$$\begin{aligned}
\min \quad & \sum_{n\in\boldsymbol{N}}\sum_{i\in\boldsymbol{A}^r[t]}W_{n,i}[t]\alpha_{n,i}[t] \\
\text{s.t.} \quad & \sum_{i\in\boldsymbol{A}^r[t]}\alpha_{n,i}[t] \le f_n^r, \forall n\in\boldsymbol{N}, \\
& \sum_{n\in\boldsymbol{N}}\sum_{i\in\boldsymbol{A}_m^r[t]}\alpha_{n,i}[t] = D_m^r[t]a_m^r[t], \forall m\in\boldsymbol{N}, \\
& \alpha_{n,i}[t]\in\{0,1\}, \forall i\in\boldsymbol{A}^r[t], n\in\boldsymbol{N}.
\end{aligned} \tag{22}$$

Clearly, if $a_m^r[t] = 0$, $\boldsymbol{A}_m^r[t] = \emptyset$; otherwise $\boldsymbol{A}_m^r[t] = \boldsymbol{D}_m^r[t]$. For each service, the maximum assignment problem (22) is reducible to the minimum cost maximum flow (MCMF) problem [31], where the constraints ensure tasks are properly offloaded. Let $\mathcal{G}_t^r = (\boldsymbol{A}^r[t] \vee \boldsymbol{N} \vee \{S, D\}, \mathcal{E})$ be the flow network graph for service r in slot t, where S, D are source and destination nodes. There are $A^r[t]$ edges connecting S to all task nodes in $\boldsymbol{A}^r[t]$. The capacity of each edge is set to 1 since every admitted task will be offloaded to the EC in current slot, which is vital for guaranteed best-effort service. There are N edges connecting all EC nodes to D with capacity $f_{n_l}^r$. For every task node $i_k \in \boldsymbol{A}^r[t]$, we add an edge from it to each EC node $n \in \boldsymbol{N}$ since tasks can either be processed in requesting ECs or offloaded to other ECs. The capacity and cost of each edge are set to 1 and $W_{n_l,i_k}[t]$. By finding the minimum cost maximum flow in $\mathcal{G}_t^r$, all type-r tasks admitted in slot t are expected to be offloaded to the ECs with the minimum objective of (22). Therefore, we can leverage the algorithms like successive shortest path (SSP) to determine the optimal inter-cloud offloading decisions in polynomial time [37].

The next is to determine optimal admission decisions such that

$$\begin{aligned}
\max \quad & \sum_n\left(\chi^r D_n^r[t]a_n^r[t] - \sum_{i\in\boldsymbol{A}_n^r[t]}\sum_{m\in\boldsymbol{N}}W_{m,i}[t]\alpha_{m,i}[t]\right) \\
\text{s.t.} \quad & a_n^r[t]\in\{0,1\}, \forall n\in\boldsymbol{N},
\end{aligned} \tag{23}$$

The average weight of decisions on offloading all tasks in $\boldsymbol{A}_n^r[t]$ is $\hat{W}_n^r[t] = \frac{1}{A_n^r[t]}\sum_{m,i}W_{m,i}[t]\alpha_{m,i}[t]$, denoting the equivalent admission cost involving workload, transmission cost and deadline urgency. (23) can thus be converted to maximizing $\sum_n(\chi^r D_n^r[t]a_n^r[t] - \hat{W}_n^r[t]D_n^r[t]a_n^r[t])$. It's a simple linear programming problem and the optimal solution reduces to a threshold-based admission rule: if $\hat{W}_n^r[t]$ is no larger than χ^r, all arrived tasks will be admitted into the system, and rejected otherwise.

2) Intra-Cloud Placement and Processing: Since intra-cloud control of different ECs are independent, we can concurrently obtain decisions on $\beta_n^r[t]$, $x_{n,i}[t]$ associated with EC n by solving

$$\begin{aligned}
\min \quad & \sum_r\left(c_n^r[t]\beta_n^r[t] - \sum_{i\in\hat{\boldsymbol{Q}}_n^r[t]}\frac{x_{n,i}[t]}{V}\left(\widetilde{Q}_n^r[t] + \widetilde{H}_n^r[t]\omega_i[t]\right)\right) \\
\text{s.t.} \quad & \sum_r\beta_n^r[t] \le S_n, \\
& \sum_{i\in\hat{\boldsymbol{Q}}_n^r[t]}x_{n,i}[t] \le \min\left\{\beta_n^r[t]f_n^r, \hat{Q}_n^r[t]\right\}, \forall r\in\boldsymbol{R}, \\
& \beta_n^r[t], x_{n,i}[t]\in\{0,1\}, \forall i\in\hat{\boldsymbol{Q}}_n^r[t], r\in\boldsymbol{R}.
\end{aligned} \tag{24}$$

Given the values of $\beta_n^r[t]$, $x_{n,i}[t]$ can be determined under their correlation constraints. If $\beta_n^r[t] = 0$, no type-r task in n gets processed. If $\beta_n^r[t] = 1$, the first f_n^r tasks in descending order of $\omega_i[t]$ will be processed if $\hat{Q}_n^r[t] > f_n^r$, otherwise all tasks get processed. Without loss of generality, we suppose all services are placed in EC n, and the corresponding processing decisions $\hat{x}_{n,i}[t]$ can be made. Let $V_n^r[t] = \sum_{i\in\hat{\boldsymbol{Q}}_n^r[t]} \frac{1}{V}(\widetilde{Q}_n^r[t] + \widetilde{H}_n^r[t]\omega_i[t])\hat{x}_{n,i}[t]$, denoting the value obtained from placement decision $\beta_n^r[t]$. Thus, (24) can be converted into the following placement problem:

$$\begin{aligned} \min \quad & \sum_r \beta_n^r[t]\left(c_n^r[t] - V_n^r[t]\right) \\ \text{s.t.} \quad & \sum_r \beta_n^r[t] \le S_n, \\ & \beta_n^r[t] \in \{0,1\}, \forall r \in \boldsymbol{R}. \end{aligned} \tag{25}$$

Every EC first locates the optimal service $r^* = \operatorname{argmin}_r(c_n^r[t] - V_n^r[t])$. There are two cases: (1) If $c_n^{r^*}[t] - V_n^{r^*}[t] > 0$, the objective function is positive for all services (i.e., large placement cost, small workload level, not urgent deadline), and it is not economical to place any service, i.e., $\beta_n^r[t] = 0, \forall r$. (2) If $c_n^{r^*}[t] - V_n^{r^*}[t] \le 0$, service r^* should be placed since current deadline urgency goes beyond EC capacity. Set $\beta_n^{r^*}[t] = 1$, $\tilde{S}_n = S_n$. Then find the second optimal service if $\tilde{S}_n = \tilde{S}_n - 1 > 0$ and perform decisions similarly, otherwise end placement process.

3) Intra-Cloud Dropping: Similarly, dropping decisions of different ECs are independent, and decisions on $y_{n,i}[t]$ for every EC $n \in \boldsymbol{N}$ can be concurrently determined by solving

$$\begin{aligned} \max \quad & \sum_{i\in\hat{\boldsymbol{Q}}_n^r[t]} y_{n,i}[t]\left(\frac{1}{V}\left(\widetilde{Q}_n^r[t] + \widetilde{H}_n^r[t]\omega_i[t]\right) - \eta^r\right) \\ \text{s.t.} \quad & \sum_{i\in\hat{\boldsymbol{Q}}_n^r[t]} (\epsilon^r - x_i[t]) y_{n,i}[t] \le y^{\max}, \\ & y_{n,i}[t] \in \{0,1\}, \forall i \in \hat{\boldsymbol{Q}}_n^r[t]. \end{aligned} \tag{26}$$

The optimal solution to (26) would prefer to make $y_{n,i}[t]$ with positive weight as large as possible. Following this intuition, each EC ranks each decision $y_{n,i}[t]$ according to the weights, and then drops each $y_{n,i}[t]$ as large as possible subject to the constraints. The remaining variables with negative weight are set to zero.

Remark. Intra-cloud dropping is compatible with processing. When task queue is idle, tasks with urgent deadlines are given priority in processing and there is no need to drop. When the queue is going overflow, try to process tasks and drop tasks approaching deadlines in view of their little service chances. By leveraging user deadline tolerance, workloads can be migrated across the slots, achieving the temporal optimality of system efficiency.

5.1.4 Performance Analysis for D-JCPS

Alg. 3 summarizes the D-JCPS implemented in each slot. For each EC, the front-end server performs inter-cloud scheduling (lines 1-8) by applying SSP to solve the pure offloading problem and on that basis, deciding whether to admit newly arrived tasks under a threshold-based policy. Meanwhile, the back-end cluster determines which services to place (line 10) and performs priority-based intra-cloud scheduling on offloaded tasks (lines 13-21). Since task dropping would not affect current control decisions, we integrate the update of $\hat{\boldsymbol{Q}}_n^r[t]$ due to the dropped tasks into the following process of updating $\boldsymbol{Q}_n^r[t]$. After that, update the dynamics of priority weight and queue backlogs (lines 22-23).

Alg. 3 includes three for loops. The first loop runs R iterations. In each iteration, apply SSP to obtain feasible offloading control

Algorithm 3: Online Mechanism D-JCPS in slot t

1 **for** *Each service* $r \in \boldsymbol{R}$ **do**
2 Apply SSP to obtain decision $\alpha_{n,i}[t]$ for $i \in \boldsymbol{A}^r[t]$;
3 **for** *Each EC* $n \in \boldsymbol{N}$ **do**
4 Derive aggregate offloading weight $\hat{W}_n^r[t]$ by (23);
5 **if** $\chi^r > \hat{W}_n^r[t]$ **then**
6 $a_n^r[t] = 1$;
7 **else**
8 $a_n^r[t] = 0$; $\boldsymbol{A}^r[t] \leftarrow \boldsymbol{A}^r[t] \backslash \boldsymbol{A}_n^r[t]$;
9 **for** *Each EC* $n \in \boldsymbol{N}$ **do**
10 Derive decision $\beta_n^r[t]$ by (25);
11 **for** *Each service* $r \in \boldsymbol{R}$ **do**
12 $\hat{\boldsymbol{Q}}_n^r[t] = \boldsymbol{Q}_n^r[t] \vee \boldsymbol{\alpha}_n^r[t]$;
13 **if** $\beta_n^r[t] = 1$ **then**
14 Derive decision $x_{n,i}[t]$ for $i \in \hat{\boldsymbol{Q}}_n^r[t]$ by (24);
15 **for** *Each task* $i \in \hat{\boldsymbol{Q}}_n^r[t]$ **do**
16 **if** $x_{n,i}[t] = 1$ **then**
17 $x_i[t] = x_i[t-1] + 1$;
18 **else**
19 $x_i[t] = x_i[t-1]$;
20 $\hat{\boldsymbol{Q}}_n^r[t] \leftarrow \hat{\boldsymbol{Q}}_n^r[t] \backslash \{i\}$ if $x_i[t] \ge \epsilon^r$;
21 Derive decision $y_{n,i}[t]$ for $i \in \hat{\boldsymbol{Q}}_n^r[t]$ by (26);
22 **for** *Each EC* $n \in \boldsymbol{N}$, *service* $r \in \boldsymbol{R}$ **do**
23 Update $Q_n^r[t]$, $Z_n^r[t]$, $\omega_i[t]$, $H_n^r[t]$ by (15)-(18).

with running time $\mathcal{O}(\min\{v_1^2 f^*, v_1^3 c^*\})$, where $v_1 = 2 + N + A^r[t]$ is the total number of vertices on $\mathcal{G}_t^r$, and f^*, c^* are the derived maximum flow and minimum cost. After that, perform admission and offloading control in $\mathcal{O}(N^2 A_n^r[t])$ steps and $\mathcal{O}(N A_n^r[t])$ steps, where $A_n^r[t] = |\boldsymbol{A}_n^r[t]|$. The second loop terminates after N iterations, in each of which perform placement decisions in $\mathcal{O}(R)$ steps and determine which tasks to process or drop in at most $\mathcal{O}(R\hat{Q}_n^r[t](2+\log(\hat{Q}_n^r[t])))$ steps. The third loop for updating takes $\mathcal{O}(NR)$ time. Thus, the overall complexity of Alg. 3 is $\mathcal{O}(R \cdot \min\{v_1^2 f^*, v_1^3 c^*\} + NR(2+(N+1)A_n^r[t] + \hat{Q}_n^r[t](2+\log(\hat{Q}_n^r[t]))))$.

We next analyze its achieved properties. Before that, we define the perturbation parameter as $\theta_n^r \triangleq 2(f_n^r + y^{\max}) + e^2\sigma H_n^{r,\max}, \forall r, n$, where $H_n^{r,\max}$ is specified later. It can be easily determined since it only requires the knowledge of maximum number of processed/dropped tasks in one slot and maximum coefficient of task urgency, without any statistical knowledge of system dynamics.

Theorem 3: *If the joint cooperative controls are done by Alg. 3 with* $Q_n^r[0] = \theta_n^r, \forall r, n$, *D-JCPS achieves the following properties: a) Let* $M = V max\{\eta^r, \frac{1}{f_n^r} c_n^{r,max}, \chi^{max}\} + \theta_n^r$. *The backlogs of task queue* Q_n^r, *incentive queue* Z_n^r *and priority queue* H_n^r *satisfy:*

$$Q_n^r[t] \le Q_n^{r,\max} \triangleq \theta_n^r + V\chi^{\max} + \epsilon^r f_n^r, \tag{27}$$

$$Z_n^r[t] \le Z_n^{r,\max} \triangleq \theta_n^r + V\chi^{\max} + (N-1)D_n^{r,\max} - E_n^r, \tag{28}$$

$$H_n^r[t] \le H_n^{r,\max} \triangleq \frac{M}{\sigma e^{2-d^{\max}}} + (e-1)\sigma Q_n^{r,\max} + e\sigma\epsilon^r f_n^r. \tag{29}$$

b) Let $Q^{\max} = \max_{r,n} Q_n^{r,\max}$, $H^{\max} = \max_{r,n} H_n^{r,\max}$. *There is no task dropping if the maximum task workload* $\epsilon^{\max} = \max_{r,n}\epsilon^r$, *the maximum processing rate* $f^{\max} = \max_{r,n} f_n^r$ *and placement decisions* $\beta_n^r[t]$ *satisfy:*

$$\begin{aligned} & NRQ^{\max}\left(e^2\sigma^2\left(e^2-1\right)H^{\max} + \left(1+e^4\sigma^2\right)\epsilon^{\max} f^{\max}\right) \\ & \le \sum_{n,r} \beta_n^r[t] f_n^r \left(1 + \sigma^2 e^{3-2d^{\max}}\right). \end{aligned} \tag{30}$$

c) Let C^* be the optimal average social cost for problem (10). The overall average social cost achieved by D-JCPS satisfies:

$$\limsup_{T\to\infty}\frac{1}{t}\sum_{t=0}^{T-1}\mathbb{E}\{C[t]\}\leq C^*+\frac{B_1}{V}. \tag{31}$$

Proof. See Appendix F. □

5.2 Auction-Based Deadline-Driven Joint Cooperative Placement and Scheduling Mechanism

The previous subsection considers a cooperative scenario where individual ECs' workload and resource cost are known for social cost minimization. Taking selfishness of ECs into account, such global optimization cannot attain without the help of an efficient market that elicits desirable behaviors from individual ECs [38]. We next design AD-JCPS, an auction-based deadline-driven cooperative control mechanism, involving inter-cloud trading and intra-cloud placement/scheduling.

5.2.1 Reverse Auction Model for Inter-Cloud Trading

Inter-cloud trading for different services is performed in parallel and we consider it only for a single service in a single slot. The same analysis can be applied to other services and slots. The reverse auction model [39] is introduced to capture the interactions between the MEC provider and ECs, where the former buying offloading services for arrived tasks is auctioneer and the latter act as bidders. For convenience, denote $\boldsymbol{D}^r[t]=\cup_{n\in\boldsymbol{N}}\boldsymbol{D}_n^r[t]$ as the set of type-r tasks arrived in slot $t\in\boldsymbol{T}$. Each offloading service is associated with a cost defined as the increase of one-shot drift-plus-cost under it. For any locally arrived task $i\in\boldsymbol{D}_n^r[t]$, the cost of EC $n\in\boldsymbol{N}$ is $c_{n,i}=\frac{1}{V}\left(\widetilde{Q}_n^r[t]+\widetilde{H}_n^r[t]\omega_i[t]\right)$. While for any task from other ECs $i\in\hat{\boldsymbol{D}}_n^r[t]=\cup_{m\in\boldsymbol{N}\setminus\{n\}}\boldsymbol{D}_m^r[t]$, $c_{n,i}=\frac{1}{V}\left(\widetilde{Q}_n^r[t]+\widetilde{H}_n^r[t]\omega_i[t]+Z_n^r[t]+V\lambda^r c_{n,n_i}[t]\right)$. The provider initiates a reverse auction by soliciting bids from ECs, where only a subset of bids are chosen for social welfare maximization.

1) Bidding: For task $i\in\boldsymbol{D}^r[t]$, every EC n submits one bid to the provider, which is specified as a 2-tuple $B_{n(i)}=(f_n^r,b_{n,i})$. Here f_n^r is the maximum number of tasks that n can offload, and $b_{n,i}$ is the claimed cost for offer offloading service to i, which may be different from the actual cost $c_{n,i}$. The set of all bids is given by $\boldsymbol{B}=\left\{B_{n(i)},\forall i\in\boldsymbol{D}^r[t],n\in\boldsymbol{N}\right\}$. On that basis, the provider determines the allocation and payment rules. Let $\pi_{n,i}$ be the binary allocation variable associated with $B_{n(i)}$, where $\pi_{n,i}=1$ represents EC n's bid on task i is successful (i.e., i is allocated to n), and $\pi_{n,i}=0$ otherwise. ECs can win a set of atomic bids, and the number of winning bids is upper bounded by f_n^r. If bid $B_{n(i)}$ wins, the provider pays n a reward $p\left(B_{n(i)},\boldsymbol{B}_{-n(i)}\right)$, where $\boldsymbol{B}_{-n(i)}$ denotes other bids in $\boldsymbol{B}$ except $B_{n(i)}$. ECs that are not allocated get no payment. The utility of EC n obtained from winning bid $B_{n(i)}$ is the difference between its payment and actual cost, i.e., $u_{n,i}=p\left(B_{n(i)},\boldsymbol{B}_{-n(i)}\right)-c_{n,i}$. For the provider, its trading utility is the difference between the service-specific value χ^r and the claimed cost of winner EC, i.e., $u_i=\chi^r-b_{n,i}$.

2) Winner Determination Problem: Winning bids are determined based on an allocation decision, with the objective of maximizing inter-cloud social welfare defined as the sum of utilities of all allocated tasks. The winner determination problem (WDP) is to find an optimal allocation $\{\pi_{n,i}^*\}$ that

$$\begin{aligned}
\max\quad & \sum_{n\in\boldsymbol{N}}\sum_{i\in\boldsymbol{D}^r[t]}\left(\chi^r-b_{n,i}\right)\pi_{n,i}\\
\text{s.t.}\quad & \sum_{i\in\boldsymbol{D}_t^r}\pi_{n,i}\leq f_n^r,\forall n\in\boldsymbol{N},\\
& \sum_{n\in\boldsymbol{N}}\pi_{n,i}\leq 1,\ \ \forall i\in\boldsymbol{D}^r[t],\\
& \text{if } \exists j\in\boldsymbol{D}^r[t],\sum_m\pi_{m,j}=0,\text{then } \pi_{n,i}=0,\forall i\in\boldsymbol{D}_{n_j}^r[t],n\in\boldsymbol{N},\\
& \pi_{n,i}\in\{0,1\},\forall i\in\boldsymbol{D}^r[t],n\in\boldsymbol{N},
\end{aligned} \tag{32}$$

where the second constraint ensures each task is only allocated once, and the third constraint suggests the decision feasibility. Take tasks requesting the same service and EC as a whole. If there exists a task unallocated, all associated tasks will be unallocated.

Lemma 4. *The objective of WDP associated with service $r\in\boldsymbol{R}$ and slot $t\in\boldsymbol{T}$ equals $\sum_n\left(\chi^r D_n^r[t]a_n^r[t]-\sum_{i\in\boldsymbol{A}^r[t]}W_{n,i}[t]\alpha_{n,i}[t]\right)$ when for every EC $n\in\boldsymbol{N}$, $\pi_{n,i}$'s correspond to $\alpha_{n,i}[t]$'s for $i\in\boldsymbol{D}^r[t]$, and $b_{n,i}=\frac{1}{V}\left(\widetilde{Q}_n^r[t]+\widetilde{H}_n^r[t]\omega_i[t]\right)$ for $i\in\boldsymbol{D}_n^r[t]$, $b_{n,i}=\frac{1}{V}\left(\widetilde{Q}_n^r[t]+\widetilde{H}_n^r[t]\omega_i[t]+Z_n^r[t]+V\lambda^r c_{n,n_i}[t]\right)$ for $i\in\hat{\boldsymbol{D}}_n^r[t]$.*

Proof. See Appendix G. □

By Lemma 4, when ECs bid truthfully about service cost, the optimal allocation decisions, $\pi_{n,i}$'s, can be derived from the optimal solution to inter-cloud scheduling problem (21), $\alpha_{n,i}[t]$'s, through the correspondence between them. It's critical to promote truthful bidding in inter-cloud trading to guarantee bidding true cost is a dominant strategy for ECs, thereby preventing gaming the system and simplifying auction design. We next propose an auction-based mechanism that enables ECs to bid truthfully.

5.2.2 Design of Auction-Based Mechanism AD-JCPS

1) Inter-Cloud Trading: AD-JCPS uses the auction-based inter-cloud trading policy to obtain the allocation decisions, which is summarized in Alg. 4. Similar to inter-cloud control in Alg. 3, it applies SSP to compute the minimum cost maximum flow (line 1) since WDP is actually a maximum assignment problem, followed by allocating tasks to ECs under a threshold-based admission policy (lines 2-6). The major difference lies in that to address the information disclosure of ECs, a VCG-based payment is introduced to motivate ECs to bid the costs truthfully (lines 7-9).

Allocation Rule: We solve WDP in the following three steps.

• **Transforming to minimum cost maximum flow problem:** We construct the auction-based flow network graph $\mathcal{G}_{t,A}^r=\left(\boldsymbol{D}_t^r\vee\boldsymbol{N}\vee\{S_A,D_A\},\mathcal{E}_A\right)$ for inter-cloud trading associated with service r and slot t. We connect source node S_A to all task nodes in $\boldsymbol{D}_t^r$ with the capacity and cost of each edge set to 1 and 0. Connect every EC node $n_l\in\boldsymbol{N}$ to destination node D_A, and the capacity and cost of the edge are $f_{n_l}^r$ and 0. We add an edge $(i_k,n_l)\in\mathcal{E}_A$ between task node i_k and EC node n_l with capacity 1 and cost $W(i_k,n_l)=b_{n,i}-\chi^r$. The value of S_A-to-D_A flow equals the sum of flows across these links, which is also the objective of (32). The desired flow is determined under the feasibility constraints in (32) such that tasks are properly allocated.

• **Computing minimum cost maximum flow:** We also employ the algorithms like SSP to compute the minimum cost maximum flow constructed on $\mathcal{G}_{t,A}^r$, f_A^*, in polynomial time.

• **Determining winning bids and task allocation:** For each flow (i_k,n_l) in f_A^*, the connection means that the bid b_{n_l,i_k} submitted by EC n_l on task i_k wins and i_k is allocated to n_l.

Algorithm 4: Inter-Cloud Trading for Service r in slot t

Apply SSP on bid set $\boldsymbol{B}$ to derive allocation $\pi_{n,i}$ by (32);
for *Each EC* $n \in \boldsymbol{N}$ **do**
  if $\sum_{i\in\boldsymbol{D}_n^r[t]}\sum_{m\in\boldsymbol{N}}\pi_{m,i} = D_n^r[t]$ **then**
    $\pi^*_{m,i} = \pi_{m,i}, \forall i \in \boldsymbol{D}_n^r[t]$;
  else
    $\pi^*_{m,i} = 0, \forall i \in \boldsymbol{D}_n^r[t]$;
for *Each EC* $n \in \boldsymbol{N}$ **do**
  if $\exists i \in \boldsymbol{D}^r[t]$ *such that* $\pi^*_{n,i} = 1$ **then**
    Derive payment $p(B_{n(i)}, \boldsymbol{B}_{-n(i)})$ by (33);

Payment Rule: As the most well-studied auction format, Vickrey-Clarke-Groves (VCG) mechanism possesses the desirable truthfulness property while yielding the socially optimal allocation [39], [40]. Since winning bids are determined based on ECs' claimed costs, VCG-based payment rule is designed to encourage ECs to truthfully report the cost. Here the payment to each winner EC equals its marginal contribution to the social welfare of other bids. The payment for winner EC n associated with task i is

$$p\left(B_{n(i)}, \boldsymbol{B}_{-n(i)}\right) = S^*\left(\boldsymbol{B}\right) - \left(-b_{n,i}\right) - S^*\left(\boldsymbol{B}_{-n(i)}\right), \quad (33)$$

where $S^*(\boldsymbol{B})$ is the maximum inter-cloud social welfare under $\boldsymbol{B}$. Such payment can be interpreted by amortizing total social welfare to each associated element in the trading, where task i has a welfare χ^r and the welfare of EC n that i is allocated is $-b_{n,i}$. The first two terms in (33) are the social welfare of all other elements except that of n when $B_{n(i)}$ wins, whose value depends on $\boldsymbol{B}_{-n(i)}$ and is irrelevant to $B_{n(i)}$. The last term is the social welfare of all other elements when excluding $B_{n(i)}$.

2) Intra-Cloud Placement and Scheduling: Once being allocated to EC n, task $i \in \boldsymbol{D}^r[t]$ will enter into the queue Q_n^r waiting for processing. The detailed process is omitted since it can be directly adapted from that of Alg. 3.

5.2.3 Performance Analysis for AD-JCPS

The following results assert the truthfulness, optimality and delay performance of AD-JCPS.

Theorem 4: *AD-JCPS is truthful, i.e., each EC reports service cost truthfully in inter-cloud trading no matter what others report.*

Proof. See Appendix H. □

Theorem 5: *AD-JCPS achieves the same queueing delay bound and asymptotically optimal social cost as D-JCPS.*

Proof. See Appendix I. □

6 Simulation

We envision an edge cloud federation system deployed in a commercial complex, where tenants deploy ECs to collaboratively serve employees. We simulate a $200m \times 200m$ commercial complex with $N = 10$ ECs distributed by a homogeneous Poisson Point Process with density of 10^{-3}, which is commonly used in previous studies [11]. The system is to provide $R = 8$ different types of services, each of which has input data size $\lambda^r \in [0.1, 0.5]Mb$ and consumes $\epsilon^r \in [1, 3]$ time slots to complete. The income of admitting each type-r task χ^r follows a uniform distribution within $[5, 10]$. For each EC, the storage capacity S_n, maximum service rate f_n^r and sharing budget E_n^r are uniformly distributed in $[1, 5]$, $[30, 35]$ and $[10, 15]$. To capture scale economy effect on infrastructure [4], we determine a basic placement cost c_n^r reversely proportional to the maximum service rate. The unit placement cost follows a Gaussian distribution with mean c_n^r and standard deviation $c_n^r/2$. The unit offloading cost between any two ECs is measured by the geographical distance. Users are randomly assigned to one of the nearby ECs. The number of arrived tasks in each slot satisfies a Poisson distribution with arrival rate uniformly distributed in $[100, 300]$. The user deadline d_i follows a uniform distribution within $[1, 4]$.

We implement the proposed mechanisms for $T = 300$ slots and compare them with three baselines: (1) **Non-Cooperative Scheduling (NCS)**: It is an enhanced version of the method from [21], where the cooperative offloading is not allowed and the workloads exceeding ECs' local capacity are dropped; (2) **Non-Cooperative Placement (NCP)**: It is inspired by the benchmark considered in [17], where ECs place services in descending order of the demands until reaching S_n regardless of what others do; (3) **Single-Slot Constraint (SSC)**: We apply the method from [10], where each EC poses a hard incentive constraint on helping peers offload type-r tasks in each slot to satisfy long-term incentive constraints, thus making no sharing budget violation of SSC across all slots. Since the above are derived from the schemes without user deadline tolerance, we extend them by simply releasing the time constraint on problem formulation for fair comparisons.

6.1 Results for Deadline-Oblivious Case

Impact of Number of ECs. Fig. 2 illustrates the average social cost with respect to the number of ECs. We observe that without cooperative offloading, NCS bears a high social cost since ECs can be easily overloaded under heterogeneous task arrival pattern, resulting in high dropping cost. Given user demands, how N changes has no effect on the overall service capacity and the performance of NCS stays roughly the same. By contrast, the other three mechanisms with cooperative offloading enabled achieve much lower social cost, and the performance improves as N increases. DO-JCPS is superior to NCP where ECs place services only based on individual service popularity. This is because by optimizing the cooperative placement and scheduling jointly, DO-JCPS can make full use of available resources and serve more tasks of different types. Moreover, DO-JCPS leverages the heterogeneity of temporal task arrival pattern, realizing more balanced workload and high cost reduction across the slots. As for SSC, the hard incentive constraints are enforced in each slot, making scheduling less flexible and thus yielding large cost.

Impact of Service Types. To capture the impact of number of service types, Fig. 3 shows the average social cost with respect to R. Compared to NCS and NCP, the joint cooperative mechanisms (SSC, DO-JCPS) achieve lower social cost, since they can efficiently utilize the available storage capacity by exploiting diversities in workload and resource cost among ECs. Interestingly, as R increases, the gap between them increases first slowly and then sharply. This is because when R is small, ECs mostly have sufficient storage capacity; as R increases, the impact of storage capacity limitation becomes more prominent. By leveraging joint cooperative placement and scheduling, such impact can be greatly mitigated for DO-JCPS. Moreover, SSC incurs a slightly higher cost than DO-JCPS due to neglect of correlations across the slots.

Spatial Load Balancing. To validate efficiency of DO-JCPS in spatial load balancing, we perform simulations on four adjacent ECs for one specific service in one specific slot. Fig. 4 depicts the distribution of initial workload and scheduled workload. Instead

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2021.3061602, IEEE Transactions on Mobile Computing

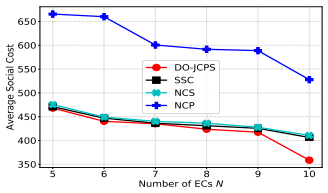


Fig. 2. Average social cost vs. N.

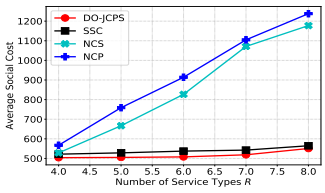


Fig. 3. Average social cost vs. R.

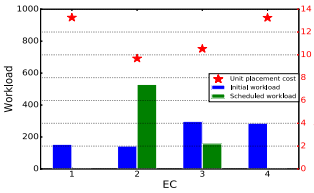


Fig. 4. Spatial load balancing.

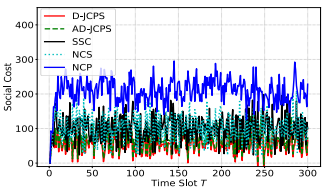


Fig. 5. Social cost vs. T.

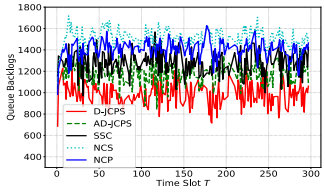


Fig. 6. Queue backlogs vs. T.

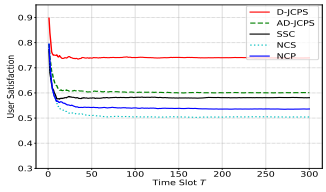


Fig. 7. Satisfaction ratio vs. T.

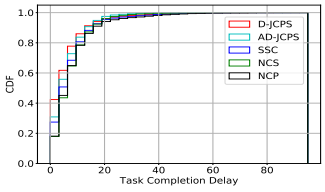


Fig. 8. CDF of completion delay.

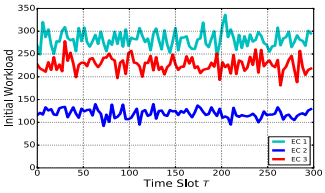


Fig. 9. Initial workload.

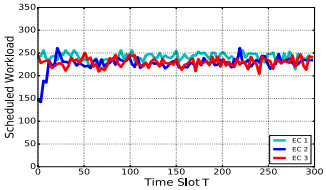


Fig. 10. Scheduled workload.

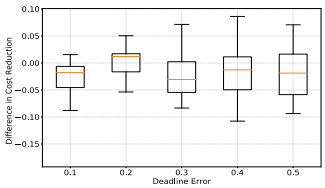


Fig. 11. Sensitivity to deadline errors.

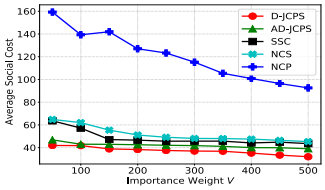


Fig. 12. Average social cost vs. V.

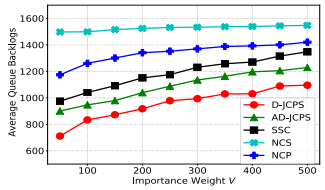


Fig. 13. Average backlogs vs. V.

TABLE 3
Average running time for deadline-oblivious mechanisms

Algorithms	Average Running Time
NCS	15.16 sec
NCP	145.31 sec
SSC	156.56 sec
DO-JCPS	162.89 sec

of simply distributing the workload evenly across ECs, DO-JCPS is designed to balance the workloads by exploiting diversities in workload and resource cost among ECs. We observe that low-cost and heavily-loaded ECs (e.g., EC 3) tend to place the service and offload tasks from high-cost and lightly-loaded ECs (e.g., EC 1), avoiding placing high-cost service just to serve less workload.

Computation Efficiency. Table 3 shows the deadline-oblivious mechanisms share different running time. Due to NP-hardness of the joint cooperative control problem, NCS and NCP that separately optimize service placement or scheduling, are faster than other solutions that consider two problems jointly. Especially for NCS, no need to interact with neighboring ECs makes it run faster. Compared to SSC, DO-JCPS is inferior because under sharing budget constraint, there exist more complex optimizations across the slots in scheduling. Taking Fig. 2 and Fig. 3 together, DO-JCPS needs longer time but for significant cost reduction.

6.2 Results for Deadline-Driven Case

Run-Time Performance. Fig. 5 and Fig. 6 show the long-term performance comparison of the deadline-driven mechanisms in terms of social cost and stability, respectively. Without cooperative offloading, NCS is expected to bear the highest social cost and largest queue backlog. As for NCP, the interplay between placement and scheduling is ignored, making adjacent ECs more likely to place the same services. This practice may lead to the waste of service capacity and incur high system cost. By contrast, the joint cooperative mechanisms (SSC, D-JCPS, AD-JCPS) allow ECs to place time-critical services at low cost and make the most of provisioned resources. We observe that the latter two that jointly optimize task scheduling across the slots are superior to SSC. This is because by posing hard sharing budget constraint in each slot, slot-by-slot optimization does not handle well the heterogeneity of temporal task arrival pattern. Moreover, AD-JCPS achieves similar performance to D-JCPS, and the slightly higher social cost and larger backlog are due to strategic behaviors of self-interested ECs.

Task Completion Performance. We mainly focus on two metrics: user satisfaction and task completion delay. User satisfaction measured by the ratio of tasks being completed before the deadlines is shown in Fig. 7. As expected, NCS achieves the lowest satisfaction since without cooperative scheduling, it is difficult for resource-limited ECs to fulfill all tasks alone. By exhausting available capacity to serve users, NCP is superior to NCS. While for SSC, the design inefficiency exists due to the sub-optimality under slot-by-slot optimization. By exploiting correlation across slots, the proposed solutions make cooperative offloading more flexible, realizing more balanced workloads in temporal domain. Fig. 8 shows the cumulative distribution function (CDF) of completion delay. The average completion delay is 3 for D-JCPS, AD-JCPS, SSC, and 6 for NCP, NCS. There is a 2-fold gap between the latter two that optimize placement and scheduling separately, and the former three that jointly consider two problems, suggesting the importance of joint cooperative control.

Spatial and Temporal Load Balancing. We validate effectiveness of the proposed D-JCPS in load balancing, taking EC 1, EC 2 and EC 3 for one specific service as an example. As shown in Fig. 9, the initial workload patterns change rapidly over time, and even within the same slot, such patterns of different ECs vary a lot. Fig. 10 illustrates the aggregate scheduled workload performance of D-JCPS. Here the aggregation involves load balancing in spatial and temporal domains, which are realized by leveraging edge cloud federation and user deadline preference. Combining these two figures, both spatial and temporal workload fluctuation can be mitigated under deadline-driven cooperative control. For example, the time-average variance of EC 1 is decreased by 77.48%.

Sensitivity to Errors in User Deadline Demands. The proposed deadline-driven mechanisms in Section 4 need to know the deadline information of user requests, which more or less has

TABLE 4
Average running time for deadline-driven mechanisms

Algorithms	Average Running Time
NCS	14.18 sec
NCP	121.79 sec
SSC	128.85 sec
D-JCPS	134.92 sec
AD-JCPS	144.06 sec

some errors in practice. We examine the sensitivity of our solutions with respect to errors in user deadline demands. For each arriving task, we particularly add a random error to the deadline length it contains, and then conduct D-JCPS on such error dataset. Using the results on datasets without errors as the baseline, we present the difference in average social cost increase due to the injected errors in user deadlines with varying error bounds (from $\pm 10\%$ of error to $\pm 50\%$ of error, uniformly distributed). As shown in Fig. 11, user deadline errors result in changes in average social cost, but only within the range of -11% to 8%. Therefore, it can be concluded that D-JCPS is robust to errors in user deadlines.

Impact of Importance Weight. Fig. 12 and Fig. 13 illustrate the impact of V on system performance. As expected, the two proposed deadline-driven joint cooperative mechanisms (D-JCPS, AD-JCPS) are superior to others (NCS, NCP, SSC). Note that larger V means emphasizing more on social cost and being less concerned with Lyapunov drift represented by queue backlogs. As V grows to the infinity, all mechanisms achieve optimal social cost and AD-JCPS achieves almost the same low social cost as D-JCPS. Combining these two figures, we observe that as V increases, a lower social cost is achieved but at the cost of larger Lyapunov drift, indicating there exists a $[O(1/V), O(V)]$ trade-off between average social cost and queue backlogs, which is consistent with our theoretical analysis in Theorem 3.

Computation Efficiency. Table 4 shows the performance comparison in running time. Similar to Table 3, NCS without inter-cloud offloading is the fastest, followed by NCP, SSC, the proposed solutions. Nevertheless, each deadline-driven mechanism incurs a lower overhead than that in deadline-oblivious case because the queued-based online control is performed to pursue asymptotically optimal solution, not the optimal one. Compared to D-JCPS, AD-JCPS needs longer time since a VCG-based payment policy is applied to elicit truthful bidding from ECs.

7 Conclusion

We study the problem of online joint cooperative placement and scheduling among federated ECs, taking both user deadline preference and ECs' strategic behavior into account. By leveraging user deadline tolerance, we harness the Lyapunov technique to exploit spatial-temporal optimality of the joint cooperative control in terms of cost reduction and load balancing for cases without and with offloading trading. Theoretical analysis and performance evaluation validate the efficiency of our mechanisms.

References

[1] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854-864, Dec. 2016.

[2] T. Ouyang, Z. Zhou, and X. Chen, "Follow Me at the Edge: Mobility-Aware Dynamic Service Placement for Mobile Edge Computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2333-2345, Oct. 2018.

[3] Y. Lin and H. Shen, "CloudFog: Leveraging Fog to Extend Cloud Gaming for Thin-Client MMOG with High Quality of Service," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 2, pp. 431-445, Feb. 2017.

[4] L. Wang, L. Jiao, J. Li, and M. Mühlhäuser, "Online Resource Allocation for Arbitrary User Mobility in Distributed Edge Clouds," in *Proc. of IEEE ICDCS*, Jun. 2017.

[5] Y. Yu, X. Bu, K. Yang, Z. Wu, and Z. Han, "Green Large-Scale Fog Computing Resource Allocation Using Joint Benders Decomposition, Dinkelbach Algorithm, ADMM, and Branch-and-Bound," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4106-4117, Jun. 2019.

[6] X. Lyu, C. Ren, W. Ni, H. Tian, and R. P. Liu, "Distributed Optimization of Collaborative Regions in Large-Scale Inhomogeneous Fog Computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 574-586, Mar. 2018.

[7] L. Tong, Y. Li, and W. Gao, "A Hierarchical Edge Cloud Architecture for Mobile Computing," in *Proc. of IEEE INFOCOM*, Apr. 2016.

[8] Y. Xiao and M. Krunz, "QoE and Power Efficiency Tradeoff for Fog Computing Networks with Fog Node Cooperation," in *Proc. of IEEE INFOCOM*, Apr. 2017.

[9] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative Mobile Edge Computing in 5G Networks: New Paradigms, Scenarios, and Challenges," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54-61, Apr. 2017.

[10] T. He, H. Khamfroush, S. Wang, T. L. Porta, and S. Stein, "It's Hard to Share: Joint Service Placement and Request Scheduling in Edge Clouds with Sharable and Non-Sharable Resources," in *Proc. of IEEE ICDCS*, Jul. 2018.

[11] L. Chen, S. Zhou, and J. Xu, "Computation Peer Offloading for Energy-Constrained Mobile Edge Computing in Small-Cell Networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619-1632, Aug. 2018.

[12] L. Chen and J. Xu, "Socially Trusted Collaborative Edge Computing in Ultra Dense Networks," in *Proc. of ACM SEC*, Oct. 2017.

[13] L. Chen, Jie Xu, S. Ren, and P. Zhou, "Spatio-Temporal Edge Service Placement: A Bandit Learning Approach," *IEEE Transactions on Computers*, vol. 17, no. 12, pp. 8388-8401, Dec. 2018.

[14] S. Pasteris, S. Wang, M. Herbster, T. He, "Service Placement with Provable Guarantees in Heterogeneous Edge Computing Systems," in *Proc. of IEEE INFOCOM*, Apr. 2019.

[15] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic Service Placement for Mobile Micro-Clouds with Predicted Future Costs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002-1016, Apr. 2017.

[16] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3702-3712, Dec. 2016.

[17] V. Farhadi, F. Mehmeti, T. He, T. L. Porta, H. Khamfroush, S. Wang, K. S Chan, "Service Placement and Request Scheduling for Data-intensive Applications in Edge Clouds," in *Proc. of IEEE INFOCOM*, Apr. 2019.

[18] K. Katsalis, T. G. Papaioannou, N. Nikaein, and L. Tassiulas, "SLA-Driven VM Scheduling in Mobile Edge Computing," in *Proc. of IEEE CLOUD*, Jun. 2016.

[19] L. Pu, X. Chen, J. Xu, and X. Fu, "D2D Fogging: An Energy-Efficient and Incentive-Aware Task Offloading Framework via Network-Assisted D2D Collaboration," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3887-3901, Dec. 2016.

[20] N. Carlsson and D. Eager, "Ephemeral Content Popularity at the Edge and Implications for On-Demand Caching," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 6, pp. 1621-1634, Jun. 2017.

[21] J. Xu, L. Chen, and P. Zhou, "Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks," in *Proc. of IEEE INFOCOM*, Apr. 2018.

[22] S. Shukla, O. Bhardwaj, A. A. Abouzeid, T. Salonidis, and T. He, "Proactive Retention-Aware Caching With Multi-Path Routing for Wireless Edge Networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1286-1299, Jun. 2018.

[23] A. -C. Pang, W. -H. Chung, T. -C. Chiu, and J. Zhang, "Latency-Driven Cooperative Task Computing in Multi-User Fog-Radio Access Networks," in *Proc. of IEEE ICDCS*, Jun. 2017.

[24] T. Li, C. S. Magurawalage, K. Wang, K. Xu, K. Yang, and H. Wang, "On Efficient Offloading Control in Cloud Radio Access Network with Mobile Edge Computing," in *Proc. of IEEE ICDCS*, Jun. 2017.

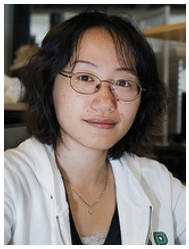
[25] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-Aware Datacenter TCP (D2TCP)," in *Proc. of ACM SIGCOMM*, Aug. 2012.

[26] Z. Zheng and N. B. Shroff, "Online Multi-Resource Allocation for Deadline Sensitive Jobs with Partial Values in the Cloud," in *Proc. of IEEE INFOCOM*, Apr. 2016.

[27] J. Zhao, H. Li, C. Wu, Z. Li, Z. Zhang, F. C.M. Lau, "Dynamic Pricing and Profit Maximization for the Cloud with Geo-distributed Data Centers," in *Proc. of IEEE INFOCOM*, Apr. 2014.
[28] R. Zhou, Z. Li, C. Wu, and Z. Huang, "An Efficient Cloud Market Mechanism for Computing Jobs With Soft Deadlines," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 793-805, Apr. 2017.
[29] Y. Yao, L. Huang, A. B. Sharma, L. Golubchik, and M. J. Neely, "Power Cost Reduction in Distributed Data Centers: A Two-Time-Scale Approach for Delay Tolerant Workloads," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 200-211, Jan. 2014.
[30] L. Wu, X. Zhang, and J. Zhang, "Capacitated Facility Location Problem with General Setup Cost," *Computers & Operations Research*, vol. 33, no. 5, pp. 1226-1241, May 2006.
[31] X. Wang, R. Jia, X. Tian, and X. Gan, "Dynamic Task Assignment in Crowdsensing with Location Awareness and Location Diversity," in *Proc. of IEEE INFOCOM*, Apr. 2018.
[32] M. Mahdian, Y. Ye, and J. Zhang, "Improved Approximation Algorithms for Metric Facility Location Problems," in *Proc. of International Workshop on Approximation Algorithms for Combinatorial Optimization*, Oct. 2002.
[33] P. Li, S. Guo, and J. Hu, "Energy-Efficient Cooperative Communications for Multimedia Applications in Multi-Channel Wireless Networks," *IEEE Transactions on Computers*, vol. 64, no. 6, pp. 1670-1679, Jun. 2015.
[34] M. H. Hajiesmaili, L. Deng, M. Chen, and Z. Li, "Incentivizing Device-to-Device Load Balancing for Cellular Networks: An Online Auction Design," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 2, pp. 265-279, Feb. 2017.
[35] M. J. Neely, "Stochastic Network Optimization with Application to Communication and Queueing Systems," Morgan & Claypool, 2010.
[36] Y. Zhao, C. -C. Liao, T. -Y. Lin, J. Yin, N. Do, C. -H. Hsu, and N. Venkatasubramanian, "SmartSource: A Mobile Q&A Middleware Powered by Crowdsourcing," in *Proc. of IEEE MDM*, May 2015.
[37] Y. Rochman, H. Levy, and E. Brosh, "Resource Placement and Assignment in Distributed Network Topologies," in *Proc. of IEEE INFOCOM*, Apr. 2013.
[38] J. Zhao, C. Wu, and Z. Li, "Cost Minimization in Multiple IaaS Clouds: A Double Auction Approach," *CoRR, vol. abs/1308.0841, Aug. 2013.*
[39] Z. Feng, Y. Zhu, Q. Zhang, H. Zhu, J. Yu, J. Cao, and L. M. Ni, "Towards Truthful Mechanisms for Mobile Crowdsourcing with Dynamic Smartphones," in *Proc. of IEEE ICDCS*, Jun. 2014.
[40] L. Gao, F. Hou, J. Huang, "Providing Long-Term Participation Incentive in Participatory Sensing," in *Proc. of IEEE INFOCOM*, Apr. 2015.
[41] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters," in *Proc. of IEEE INFOCOM*, Apr. 2013.

Yuqing Li received the B.S. degree in Communication Engineering from Xidian University, Xi'an, China, in 2014, and is currently pursuing the Ph.D. degree in Electronic Engineering at Shanghai Jiao Tong University, Shanghai, China. Her current research interests include edge/mobile computing, social aware networks, privacy/security and network economics.

Wenkuan Dai received the B.S. degree in Communication Engineering from Xidian University, Xi'an, China, in 2016. He is pursuing the M.S. degree from the Department of Electronic Engineering, Shanghai Jiao Tong University. His research interests include charging station planning, and electric vehicles grid integration.

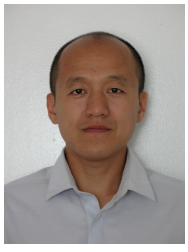


Xiaoying Gan received her Ph. D degrees in Electronic Engineering from Shanghai Jiao Tong University, Shanghai, China in 2006. She is currently with Institute of Wireless Communication Technology, at Department of Electronic Engineering, Shanghai Jiao Tong University, where she is an Associate Professor. Her current research interests include Heterogenous Cellular Network, Cognitive Network, Multi-user Multi-channel Access and Dynamic Radio Resource Management.

Haiming Jin received the B.S. degree from Shanghai Jiao Tong University, Shanghai, China, in 2012, and the Ph.D. degree from the University of Illinois at UrbanaChampaign (UIUC), Urbana, IL, USA, in 2017. He is currently a tenure-track Assistant Professor at the John Hopcroft Center for Computer Science and the Department of Electronic Engineering, Shanghai Jiao Tong University. Before this, he was a Post-Doctoral Research Associate with the Coordinated Science Laboratory, UIUC. His research of interests are in the area of urban computing, crowd and social sensing systems, network economics and game theory, reinforcement learning.

Luoyi Fu received her B. E. degree in Electronic Engineering from Shanghai Jiao Tong University, China, in 2009 and Ph.D. degree in Computer Science and Engineering in the same university in 2015. She is currently an Assistant Professor in Department of Computer Science and Engineering in Shanghai Jiao Tong University. Her research of interests are in the area of social networking and big data, scaling laws analysis in wireless networks and random graphs.

Huadong Ma received his Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 1995. He is currently a Chang Jiang Scholar Professor with the Beijing University of Posts and Telecommunications, Beijing, where he is also the Director of the Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia and the Executive Dean of the School of Computer Science. His research interests include multimedia system and networking, sensor networks, and Internet of Things.



Xinbing Wang (SM'12) received the B.S. degree (with hons.) in automation from Shanghai Jiao Tong University, Shanghai, China, in 1998, the M.S. degree in computer science and technology from Tsinghua University, Beijing, China, in 2001, and the Ph.D. degree with a major in electrical and computer engineering and minor in mathematics from North Carolina State University, Raleigh, in 2006. Currently, he is a professor with the Department of Electronic Engineering, Shanghai Jiao Tong University. His research interests include resource allocation in mobile and wireless networks, and congestion control over wireless ad hoc and sensor networks.