Energy-Efficient Real-Time Job Mapping and Resource Management in Mobile-Edge Computing

Chuanchao Gao[1,2] , Niraj Kumar[1], Arvind Easwaran[1,2]
[1]*College of Computing and Data Science*
[2]*Energy Research Institute @ NTU, Interdisciplinary Graduate Programme*
Nanyang Technological University, Singapore
gaoc0008@e.ntu.edu.sg, iraj.kumar@ntu.edu.sg, arvinde@ntu.edu.sg

***Abstract*—Mobile-edge computing (MEC) has emerged as a promising paradigm for enabling Internet of Things (IoT) devices to handle computation-intensive jobs. Due to the imperfect parallelization of algorithms for job processing on servers and the impact of IoT device mobility on data communication quality in wireless networks, it is crucial to jointly consider server resource allocation and IoT device mobility during job scheduling to fully benefit from MEC, which is often overlooked in existing studies. By jointly considering job scheduling, server resource allocation, and IoT device mobility, we investigate the deadline-constrained job offloading and resource management problem in MEC with both communication and computation contentions, aiming to maximize the total energy saved for IoT devices. For the offline version of the problem, where job information is known in advance, we formulate it as an Integer Linear Programming problem and propose an approximation algorithm, `LHJS`, with a constant performance guarantee. For the online version, where job information is only known upon release, we propose a heuristic algorithm, `LBS`, that is invoked whenever a job is released. Finally, we conduct experiments with parameters from real-world applications to evaluate their performance.**

***Index Terms*—Mobile-Edge Computing, Job Offloading and Scheduling with Deadlines, Approximation Algorithm**

I. Introduction

Internet of Things (IoT) has emerged as a forefront technology (Cisco predicts 500 billion IoT devices by 2030 [1]) driven by advancements in hardware, software, and communication technologies [2] such as low-power wide-area networks, WiFi, and ultra-reliable low-latency communication of 5G. Concurrently, the evolution of Artificial Intelligence has led to many computation-intensive IoT applications such as virtual/augmented reality [3], image/video processing [4], and object detection in autonomous driving [5]. These applications, many with rigid timing constraints, pose significant challenges to IoT devices, which are typically battery-powered and resource-constrained for compactness and portability [6].

Mobile-Edge Computing (MEC) has emerged as a promising paradigm enabling IoT devices to effectively support computation-intensive and time-critical applications. In MEC, jobs are offloaded by end devices (EDs) to nearby access points (APs) through wireless networks and then forwarded to servers via a wired backhaul network for processing. Deploying servers close to EDs in MEC significantly reduces communication latency compared to cloud computing, enabling prompt responses to ED requests. Offloading computation-intensive jobs to servers conserves EDs' energy and accelerates job processing, but also introduces additional latency and energy consumption for job offloading. Furthermore, considering the limited communication and computation resources of APs and servers, respectively, efficient job mapping (to APs and servers) and resource management (for offloading, processing, and downloading) strategies in MEC become crucial, especially for time-critical applications.

Resource management comprises two tasks: resource allocation and job scheduling. Resource allocation is concerned with the problem of how much resource to allocate to a job, i.e., in terms of amount of resource units (e.g., number of processing cores on servers). Job scheduling, on the other hand, is concerned with the problem of when to schedule the allocated resources for the jobs so as to meet job deadlines.

In this study, we focus on the problem of job mapping and resource management for deadline-constrained jobs with the following considerations. For many applications, including those that utilize GPUs, resource utilization efficiency decreases as computation parallelism increases due to the imperfect parallelization of algorithms within an application. Always allocating full server resources to each job can lead to server resources being underutilized, highlighting the importance of considering resource allocations during scheduling. Orthogonally, the wireless network condition varies with ED mobility. Considering ED mobility during job scheduling can save energy for EDs, for example, by scheduling job offloading when the network condition is optimal between EDs and APs. Although some studies have explored the problem of job mapping and scheduling for deadline-constrained jobs in MEC [7]–[27], to our best knowledge, the joint consideration of job mapping, resource management (including allocations) and ED mobility is absent.

This study addresses a deadline-constrained job mapping and resource management problem in MEC with both communication and computation contentions, aiming to maximize the total energy that can be saved for EDs; we refer to it as the Energy Maximization Job Scheduling Problem (`EMJS`). Offloaded jobs compete with each other for both wireless bandwidth on APs and computation resource on servers. We jointly consider computation resource allocation on servers,

This work was supported in part by the MoE Tier-2 grant MOE-T2EP20221-0006.

job scheduling (for offloading, processing, and downloading) on APs and servers and ED mobility. Our model incorporates a versatile job mapping framework, enabling each job to be offloaded to any of its accessible APs and subsequently forwarded to one of the capable servers for processing. Upon completion, the job (result) is relayed back to any of its accessible APs and subsequently downloaded to its ED. Considering ED mobility, the accessible APs for offloading may differ from those for downloading. Additionally, each job can only be processed on servers possessing the resource type demanded by the job. Each server offers multiple predefined computation resource allocation options; each job can be allocated any one of these options resulting in different processing durations. Notably, in scenarios where MEC comprises only one AP and a co-located server with each job always being allocated full server resources, EMJS is equivalent to a three-machine flow shop problem [28], known as NP-Hard. Hence, considering options for computation resource allocation and MECs with several APs and servers, the general EMJS is also NP-Hard.

We consider two versions of EMJS in this paper: online and offline. In online EMJS, information about each job is only available upon its release in the system. In contrast, in offline EMJS, information about all the jobs is available apriori. To address offline EMJS, we first formulate it as an Integer Linear Programming (ILP) problem by enumerating all possible *schedule instances*, where a schedule instance is a combination of (i) mapping of jobs to APs and servers, (ii) computation resource allocation on servers, and (iii) starting times for offloading, processing, and downloading (*jobs are assumed to be scheduled non-preemptively on each resource*). Then, we present a pseudo-polynomial approximation algorithm, called Light-Heavy Job Scheduling (LHJS), with a proven constant approximation ratio. For online EMJS, we propose a heuristic algorithm, called Load Balanced Job Scheduling (LBS), that is invoked whenever a job is released in the system. The contributions of this work are summarized below.

- We address the EMJS in MEC with both communication and computation contentions, aiming to maximize the total saved energy for EDs. We jointly consider job mapping, resource management and ED mobility.
- We formulate the offline EMJS as an ILP problem by enumerating all schedule instances. Then, we propose, to the best of our knowledge, a first approximation algorithm (LHJS) for EMJS with a constant approximation ratio. LHJS divides all instances into two sets based on computation resource allocation and schedules them separately.
- For the online version of EMJS, we present a heuristic scheduling algorithm called LBS by scheduling each job on the least busy server and allocating the minimum feasible computation resource to each job.
- We experimentally evaluate LHJS and LBS using profiled data of real-world applications. Results show that LHJS outperforms its baseline algorithm [18] by 11.8% in offline scheduling, and LBS achieves an average of 83.2% of the optimal saved energy in online scheduling.

II. Related Work

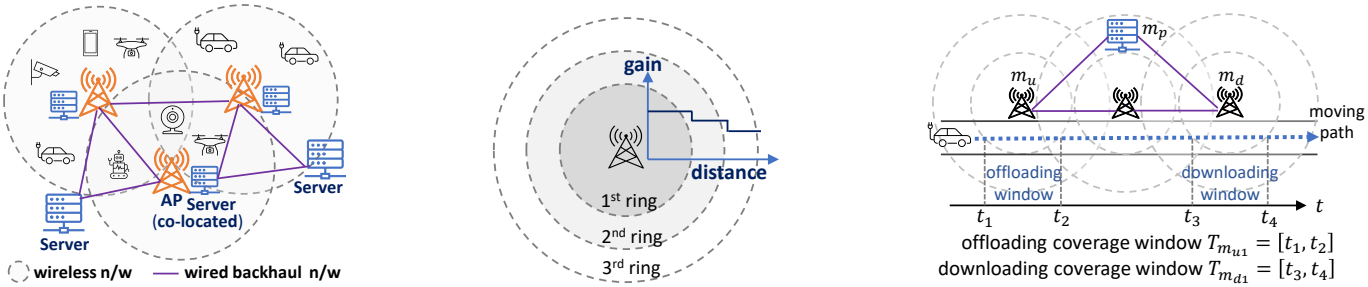
Due to the promising prospects of MEC, research interest in job mapping and resource management has grown significantly. Readers seeking a comprehensive overview of this topic can refer to related surveys [29]–[31]. Research problems can be categorized into deadline-constrained problems and deadline-free (or response time minimization) problems. This section reviews state-of-the-art research on *deadline-constrained* job mapping and *resource management* in MEC.

Depending on whether resource contention is taken into consideration (either computation or communication), these studies can be categorized into those (i) with no resource contention [7]–[9], (ii) only computation contention [10]–[17] or only communication contention [18]–[20], and (iii) both communication and computation contentions [21]–[27].

Some studies [10], [12], [14] have jointly considered job scheduling and computation resource allocation. These studies considered deploying services or virtual machines (VMs) on servers for job processing, where multiple services or VMs could run on the same server simultaneously, provided the total allocated resources did not exceed the server's resource capacity. Jobs were then mapped to and processed by each service or VM. Under this setup, all jobs mapped to the same service or VM are allocated the same amount of computation resource. Moreover, Gao *et al.* [14] assigned equal amount of computation resources to a fixed number of VMs for each server. Unlike these studies, our research considers resource allocations for each job separately, i.e., a job mapped to a server can be allocated with any of the offered computation resource allocation options. Furthermore, *none of the above studies account for communication contention for job offloading/downloading or ED mobility*.

Some studies [11], [16], [18], [20] have considered ED mobility in MEC. These studies assumed that ED trajectories were predictable, allowing the determination of accessible APs for jobs and the corresponding network channel status. Sang *et al.* [16] assumed that channel noise was a function of the physical distance between EDs and APs. Sorkhoh *et al.* [11] and Zhu *et al.* [18] divided each wireless network into several ranges such that the channel gain (or data rate) within each range remained constant, thereby reducing the accuracy requirement for trajectory prediction; in our study, we consider a similar mobility model. However, unlike our work, *none of these studies considered computation resource allocation during job scheduling*. Furthermore, Sorkhoh *et al.* [11] and Sang *et al.* [16] did not consider communication contention, while Zhu *et al.* [18] and Huang and Yu [20] did not account for computation resource contention. Moreover, Zhu *et al.* [18] proposed an offline scheduling algorithm with a parameterized approximation bound, which could be applied to our problem after some modifications (described in Section VI-A); thus, the modified algorithm of Zhu *et al.* is used as the offline baseline algorithm in evaluating LHJS in our experiments.

Existing solutions to deadline-constrained job mapping and resource management problems can be classified into heuris-



(a) (b) (c)

Fig. 1. (a) a mobile edge computing system; (b) partition the coverage area of a vAP into three rings; (c) an example of an offloaded job. A job generated by a vehicle is offloaded to vAP m_u when covered by network ring m_{u1}, forwarded to server m_p, processed on m_p, (result) forwarded to vAP m_d, and (result) downloaded from m_d when covered by network ring m_{d1}.

tic algorithms [7]–[9], [11]–[26] and exponential-time exact algorithms [10], [27]. Among heuristic algorithms, approximation algorithms [14], [18], [21] offer additional theoretical guarantees. Unlike our work, Gao *et al.* [14] did not consider communication contention, and Meng *et al.* [21] did not address the computation resource allocation during job scheduling and only derived parameterized approximation bounds. Furthermore, both studies did not consider ED mobility.

Our study jointly considers job mapping, resource management (including computation resource allocation) and ED mobility in MEC with both communication and computation contentions. Furthermore, we propose the first approximation algorithm with a constant approximation ratio for this problem.

III. System Model and Problem Formulation

A. Mobile Edge Computing System Architecture

A MEC comprises EDs, APs, and servers (Fig. 1(a)). A job can be offloaded by its ED to one of its accessible APs via a wireless network, and then forwarded to one of its capable servers (i.e., servers that possess the resource type demanded by the job) for processing through the backhaul network. After processing, the job (result) is relayed to one of its accessible APs and subsequently downloaded from the AP to its ED. Due to ED mobility, the set of accessible APs of a job may change over time so that APs used for offloading and downloading may be different. In this paper, *we use discrete time, so that all the temporal parameters are specified as integers*. A notation summary is provided in Table I.

In a wireless network, the uplink and downlink channels of each AP are usually separated by a guard band, facilitating simultaneous data offloading and downloading within the same wireless network without interference. For analytical simplicity, we define two virtual APs, referred to as vAPs, corresponding to each AP such that each vAP exclusively contains one uplink or downlink channel. Let $\mathcal{M}^u$ be the set of M^u vAPs with uplink channels, and $\mathcal{M}^d$ be the set of M^d vAPs with downlink channels. Since the channel gain of wireless networks decreases with the increasing physical distance between EDs and vAPs [18], for each vAP, we divide its wireless network coverage area into several *rings* such that the channel gains remain unchanged (from a practical viewpoint) within the same ring (Fig. 1(b)); we use m_{ir} to denote the r-th ring of vAP $m_i \in \mathcal{M}^u \cup \mathcal{M}^d$. In this paper, we consider heterogeneous servers that provide different types of computation resources, and a job can only be processed on a server possessing the resource type demanded by the job. Let $\mathcal{M}^p$ be the set of M^p servers.

We refer to a vAP or server in MEC as a *machine*, and let $\mathcal{M} = \mathcal{M}^u \cup \mathcal{M}^p \cup \mathcal{M}^d$ be the set of all machines in MEC, and let $M = \max\{M^u, M^p, M^d\}$. Let α_i denote the resource capacity of a machine $m_i \in \mathcal{M}$. The bandwidth capacity of a vAP is measured in MegaHertz (MHz). We consider Orthogonal Frequency Division Multiplexing (OFDM) as the network access method for EDs, which has been widely used in modern wireless networks such as WiFi-5 [32], 4G-LTE [33], and 5G [34]. In OFDM-enabled networks, each channel uses multiple overlapping but orthogonal subcarriers for data transmission, and EDs are distinguished based on time segments (in each time segment, one ED occupies all subcarriers in a channel to transmit a complete data packet [35]), resulting in *sequential and non-preemptive data transmission in the wireless network*. The computation resource capacity of a server is measured in computing units which are defined based on the resource type supported by the server, i.e., the computing unit of a CPU server is the CPU core, and that of a GPU server is the streaming multiprocessor. Each server $m_i \in \mathcal{M}^p$ comprises exactly one resource type (e.g., CPU, GPU, etc.) and provides a set of resource allocation options $\mathcal{C}_i$ for job processing, where $|\mathcal{C}_i| \leq C$ for some integer C, and each option $c \in \mathcal{C}_i$ corresponds to some fraction of α_i, i.e., $c \in (0, 1]$. Notably, servers with multiple resource types can be viewed as multiple co-located servers, each possessing a single resource type. Given the potentially significant context switch overhead [36], we consider *non-preemptive job processing*. Besides, servers can run multiple jobs concurrently, provided their combined resource allocations do not exceed the server's capacity.

Let $\mathcal{N}$ be the set of N jobs to be processed. Each job $n_j \in \mathcal{N}$ is represented by $\langle \theta_j^{in}, \theta_j^{out}, \gamma_j, \delta_j, \mathcal{M}_j^p, \psi_j \rangle$. Here, θ_j^{in} is the input data size, θ_j^{out} is the output data size, γ_j is the release time, and δ_j is the absolute deadline of job n_j. Both θ_j^{in} and θ_j^{out} are measured in Megabyte (MB). Each job n_j can be processed on a server with the required resource

type; let $\mathcal{M}_j^p$ be the set of servers where n_j can be processed. Due to the design of network rings, the accuracy requirement for trajectory estimation is reduced (i.e., we only need to determine if vehicles are under the coverage of network rings); thus, the ED trajectories can be estimated by mainstream navigation applications such as Google Maps [37]. Based on the estimated trajectory of n_j's ED, the accessible vAPs of n_j can be defined by a set of at most R *ring coverage windows* ψ_j throughout its lifetime $[\gamma_j, \delta_j]$. Each coverage window $T_{ir} \in \psi_j$ indicates that n_j's ED is covered by network ring m_{ir} during time window T_{ir}; the starting and ending time slots of T_{ir}, termed $st(T_{ir})$ and $et(T_{ir})$, depend on the estimated moving speed and direction of n_j's ED. If m_{ir} is chosen for offloading or downloading, we require that the operation must be completed within T_{ir}. We provide an example of the full cycle of an offloaded job in Fig. 1(c). Each job can either be processed locally or on servers. We use d_j^{loc} to denote the local processing duration of job n_j, and d_{jic} to denote the processing duration of job n_j when it is processed on server m_i with resource allocation option $c \in \mathcal{C}_i$. We assume that the ED of each job n_j does not generate other jobs during $[\gamma_j, \delta_j]$ and $\gamma_j + d_j^{loc} - 1 \leq \delta_j$ (ensuring local processing feasibility).

Enabling jobs to be forwarded to different servers via the wired backhaul network after being offloaded offers advantages in balancing server workloads and mitigating wireless network coverage limitations. For instance, the servers co-located with the accessible vAPs of job n_j may not possess the resource type required by n_j. We consider the *backhaul network* connects vAPs and servers with optical cables and is enabled with Software Defined Network technology [10]. Such a backhaul network has enough bandwidth capacity [38] to support data transmission with no communication contention and provides a fixed data transmission rate for each link. Then, given the topology of the backhaul network, the data forwarding duration between a vAP m_i and a server m_p can be defined as a function of the transmitted data size θ. It comprises the data transmission duration over each link of the shortest path from vAP m_i to server m_p; we denote this function as $\beta_{ip}(\theta)$. Note that $\beta_{ip}(\theta) = \beta_{pi}(\theta)$ and $\beta_{ii}(\theta) = 0$.

B. Energy Consumption and Timing Model

1) Local Processing: When job n_j is processed locally, the processing duration is d_j^{loc}. Let p_j^{loc} be the processing power. The energy consumption for processing job n_j locally is then given by $e_j^{loc} = p_j^{loc} \cdot d_j^{loc}$.

2) Remote Processing: Suppose job n_j starts offloading to vAP m_u at time t_j^u during ring coverage window T_{ur}. Then, it is forwarded to server m_p and starts processing at time t_j^p with resource allocation $c \in \mathcal{C}_p$. After processing, job n_j (result) is forwarded to vAP m_d and starts downloading from m_d to n_j's ED at time t_j^d during ring coverage window T_{ds}. Here, the *subscript* u, p, and d (e.g., m_u) are used as machine index, and the *superscript* u, p, and d (e.g., t_j^u) are used to denote offloading, processing, and downloading.

To ensure machine (vAP and server) capability, we have

$$T_{ur} \in \psi_j, T_{ds} \in \psi_j, \text{ and } m_p \in \mathcal{M}_j^p. \tag{1}$$

TABLE I
NOTATION (MAIN PARAMETERS AND VARIABLES)

Notation	Definition
$\mathcal{M}^u$	the set of M^u vAPs with uplink channels
$\mathcal{M}^p$	the set of M^p servers
$\mathcal{M}^d$	the set of M^d vAPs with downlink channels
$\mathcal{M}$	$\mathcal{M} = \mathcal{M}^u \cup \mathcal{M}^p \cup \mathcal{M}^d$, set of machines (vAPs and servers); $m_i \in \mathcal{M}$ denotes a machine; $M = \max\{M^u, M^p, M^d\}$
α_i	the resource capacity of machine $m_i \in \mathcal{M}$
$\mathcal{C}_i$	resource allocation options of server $m_i \in \mathcal{M}^p$; $\lvert\mathcal{C}_i\rvert \leq C$
m_{ir}	r-th network ring of vAP $m_i \in \mathcal{M}^u \cup \mathcal{M}^d$
$\mathcal{N}$	the set of N jobs, where $n_j \in \mathcal{N}$ denotes a job
θ_j^{in}	input data size of job n_j
θ_j^{out}	output (result) data size of job n_j
γ_j	release time of job n_j
δ_j	absolute deadline of job n_j
Δ	$\Delta = \max_{n_j \in \mathcal{N}} \delta_j$, total number of time units for scheduling
$\mathcal{M}_j^p$	set of servers on which job n_j can be processed; $\mathcal{M}_j^p \subseteq \mathcal{M}^p$
ψ_j	$\{T_{ir}, ...\}$, set of ring coverage windows of job n_j; $\lvert\psi_j\rvert \leq R$
t_j^u	the offloading starting time of job n_j
t_j^p	the processing starting time of job n_j
t_j^d	the downloading starting time of job n_j
d_j^u	offloading duration of job n_j
d_j^p	processing duration of job n_j
d_j^d	downloading duration of job n_j
$d_j^{u,p}$	forwarding duration of job n_j's input from its offloading vAP to its processing server
$d_j^{p,d}$	forwarding duration of job n_j's output from its processing server to its downloading vAP
e_j^{save}	saved energy of job n_j's ED by processing n_j on a server
ℓ	$\ell \triangleq \langle m_\ell^u, m_\ell^p, m_\ell^d, I_\ell^u, I_\ell^p, I_\ell^d, c_\ell^p \rangle$ is a schedule instance. m_ℓ^u, m_ℓ^p and m_ℓ^d are job mappings, I_ℓ^u, I_ℓ^p and I_ℓ^d are operation intervals, and c_ℓ^p is allocated computation resource
$e(\ell)$	saved energy of schedule instance ℓ
$\mathcal{L}$	the set of all schedule instances of all jobs
$\mathcal{L}_j$	the set of schedule instances of j
$st(T)$	start time of time interval/window T
$et(T)$	end time of time interval/window T
$x(\ell)$	binary selection variable of instance ℓ

Since job n_j cannot start offloading before its release,

$$\gamma_j \leq t_j^u. \tag{2}$$

Based on Shannon's theorem [39], the data offloading rate is defined as $\eta_j^u = \alpha_u \cdot \log_2(1 + p_j^u \cdot h_{ur}/\sigma^2)$, where α_u is the bandwidth capacity of vAP m_u, p_j^u is the offloading power of ED, h_{ur} is the channel gain in network ring m_{ur}, and σ is the noise spectral density. The offloading duration d_j^u and energy consumption e_j^u are given by $d_j^u = \theta_j^{in}/\eta_j^u$ and $e_j^u = p_j^u \cdot d_j^u$. Furthermore, t_j^u needs to satisfy

$$st(T_{ur}) \leq t_j^u, \text{ and } t_j^u + d_j^u - 1 \leq et(T_{ur}). \tag{3}$$

Since there is no bandwidth contention in the backhaul network, job n_j can be immediately forwarded to server m_p after being offloaded. Let $d_j^{u,p} = \beta_{up}(\theta_j^{in})$ be job n_j's forwarding duration from vAP m_u to server m_p. As job n_j cannot start processing before it arrives m_p, t_j^p needs to satisfy

$$t_j^u + d_j^u + d_j^{u,p} \leq t_j^p. \tag{4}$$

Let $d_j^p = d_{jpc}$ denote the processing duration of job n_j on server m_p. Once job n_j is processed, the result can

be immediately forwarded to its downloading vAP m_d. Let $d_j^{p,d} = \beta_{pd}(\theta_j^{out})$ denote this forwarding duration from server m_p to vAP m_d. Job n_j cannot start downloading from vAP m_d before it arrives at m_d, and hence t_j^d needs to satisfy

$$t_j^p + d_j^p + d_j^{p,d} \leq t_j^d. \tag{5}$$

Each ED requires a minimum power to ensure the necessary sensitivity to *receive* the wireless signal sent from a vAP [40]. Let p_j^d be the downloading power of job n_j's ED. Let $\eta_j^d = \alpha_d \cdot \log_2(1 + SNR_d)$ be the data downloading rate of n_j, where SNR_d is the signal-to-noise ratio and depends on vAP m_d since it is the data transmitter. Thus, the downloading duration d_j^d and energy consumption e_j^d for n_j are given by $d_j^d = \theta_j^{out}/\eta_j^d$ and $e_j^d = p_j^d \cdot d_j^d$. Since n_j can only be downloaded during ring coverage window T_{ds}, t_j^d must satisfy

$$st(T_{ds}) \leq t_j^d, \text{ and } t_j^d + d_j^d - 1 \leq et(T_{ds}). \tag{6}$$

To meet the job deadline, t_j^d also needs to satisfy

$$t_j^d + d_j^d - 1 \leq \delta_j. \tag{7}$$

We omit the vAP handover delay for switching from the offloading vAP to the downloading vAP (if they are different), as this delay is typically shorter than the job processing time (i.e., the vAP handover can be completed before the job processing finishes). The saved energy for job n_j's ED by processing job n_j on server m_p is defined as $e_j^{save} = e_j^{loc} - e_j^u - e_j^d$.

C. Problem Formulation

Problem Definition: Let $\Delta = \max_{n_j \in \mathcal{N}} \delta_j$. This paper aims to find a schedule that satisfies machine capability constraints, resource capacity constraints, and timing constraints for all jobs over the Δ time units to maximize the total saved energy for EDs; we refer to this problem as the Energy Maximization Job Scheduling Problem (EMJS). To solve EMJS, we need to determine the job mapping (where each job is offloaded, processed and downloaded), computation resource allocation on server, and the starting times for each *operation* (offloading, processing, and downloading). The *machine capability constraint* is specified by Eqs. (1), (3) and (6). The *resource capacity constraint* ensures that at any given time instant, the total resource allocated to all the jobs by any machine does not exceed its capacity. The *timing constraints* are specified by Eqs. (2), (4), (5), and (7).

Problem Complexity: When job executions are sequential on servers (no fractional resource allocation) and the MEC comprises one offloading vAP, one server and one downloading vAP, the resulting problem is a *three-machine flow shop problem*, known to be NP-Hard and challenging to be approximated within $\mathcal{O}(3/\log 3)$ unless P = NP [28]. Since EMJS considers a more general MEC and multiple computation resource allocation options, it is at least as hard as the three-machine flow shop problem.

Definition 1 (Schedule Instance)**.** *A schedule instance of job n_j is defined as $\ell \triangleq \langle m_\ell^u, m_\ell^p, m_\ell^d, I_\ell^u, I_\ell^p, I_\ell^d, c_\ell^p \rangle$, which represents the scenario where job n_j offloads to vAP m_ℓ^u in the operation interval I_ℓ^u ($I_\ell^u \triangleq [t_j^u, t_j^u + d_j^u - 1]$), executes on server m_ℓ^p with computation resource allocation $c_\ell^p \in \mathcal{C}_\ell$ in the operation interval I_ℓ^p ($I_\ell^p \triangleq [t_j^p, t_j^p + d_j^p - 1]$), and downloads its result from vAP m_ℓ^d in the operation interval I_ℓ^d ($I_\ell^d \triangleq [t_j^d, t_j^d + d_j^d - 1]$). Moreover, ℓ needs to satisfy*

(i) machine capability constraint: Eqs. (1)*,* (3)*, and* (6)*.*

(ii) timing constraint: Eqs. (2)*,* (4)*,* (5)*, and* (7)*.*

For ease of presentation, we use notation I_ℓ^* (likewise m_ℓ^*) to generically denote any of the three operation intervals (likewise machines) of ℓ.

Let ℓ be a schedule instance of job n_j. Let $e(\ell) = e_j^{save}$ be the *saved energy* for n_j's ED when n_j is scheduled following ℓ. For any set of schedule instances $\mathcal{S}$, let $e(\mathcal{S}) = \sum_{\ell \in \mathcal{S}} e(\ell)$. For any machine m_i and time instant t, let $\mathbb{1}_i(t, I_\ell^*)$ be an indicator function, where $\mathbb{1}_i(t, I_\ell^*) = 1$ if and only if the operation interval I_ℓ^* is active at time t ($t \in I_\ell^*$) and $m_i = m_\ell^*$.

In offline EMJS, information about all the jobs is available apriori. Thus, we can enumerate all possible schedule instances for each job $n_j \in \mathcal{N}$. Let $\mathcal{L}$ be the set of all possible schedule instances for all the jobs in $\mathcal{N}$, and $\mathcal{L}_j$ be the set of all possible schedule instances for any job n_j. Note that we are only interested in those schedule instances ℓ with positive $e(\ell)$; thus, $\mathcal{L}$ and $\mathcal{L}_j$ contain only those schedule instances with positive saved energy. Since each job has at most R ring coverage windows and each server has at most C resource allocation options, we have at most $NMCR^2\Delta^3$ schedule instances in $\mathcal{L}$. Because Δ depends on the value of δ_j, the number of schedule instances in $\mathcal{L}$ is pseudo-polynomial to the input size. For each schedule instance $\ell \in \mathcal{L}$, let $x_\ell \in \{0, 1\}$ be the *selection variable* of ℓ, where $x_\ell = 1$ if and only if ℓ is selected in the solution. Then, we formulate offline EMJS as follows.

$$\text{(EMJS Offline)} \quad \max \sum_{\ell \in \mathcal{L}} e(\ell) \cdot x_\ell \tag{8}$$

subject to:

$$\sum_{\substack{\ell \in \mathcal{L}, I_\ell^* \in \ell, \\ \mathbb{1}_i(t, I_\ell^*) = 1}} x_\ell \leq 1, \forall 1 \leq t \leq \Delta, \forall m_i \in \mathcal{M}^u \cup \mathcal{M}^d \tag{8a}$$

$$\sum_{\ell \in \mathcal{L}, \mathbb{1}_i(t, I_\ell^p) = 1} x_\ell \cdot c_\ell^p \leq 1, \forall 1 \leq t \leq \Delta, \forall m_i \in \mathcal{M}^p \tag{8b}$$

$$\sum_{\ell \in \mathcal{L}_j} x_\ell \leq 1, \quad \forall n_j \in \mathcal{N} \tag{8c}$$

$$x_\ell \in \{0, 1\}, \quad \forall \ell \in \mathcal{L} \tag{8d}$$

Eqs. (8a) and (8b) are the resource capacity constraints for vAPs and servers, respectively. Constraint (8c) guarantees that at most one schedule instance of each job is selected in the solution. Note that the machine capability and timing constraints have already been considered in the definition of schedule instances. The formulation contains $NMCR^2\Delta^3$ variables and $(3M\Delta + J)$ constraints, which is a pseudo-polynomial of the input size.

IV. An Approximation Algorithm for Offline EMJS

This section presents an approximation algorithm for offline EMJS, called the Light-Heavy Job Scheduling Algorithm (LHJS). (For ease of presentation, whenever we mention EMJS in this section it always refers to the offline version.) For each schedule instance $\ell \triangleq \langle m_\ell^u, m_\ell^p, m_\ell^d, I_\ell^u, I_\ell^p, I_\ell^d, c_\ell^p \rangle$ of job n_j, $0 < c_\ell^p \leq 1$ denotes the computation resource allocation on server m_ℓ^p. We refer to ℓ as a *light schedule instance* if $c_\ell^p \leq \frac{1}{2}$, and as a *heavy schedule instance* otherwise. Further, for ease of presentation, we use c_ℓ^* to generically denote the resource allocation corresponding to any operation interval I_ℓ^* in ℓ; if $I_\ell^* = I_\ell^p$ then $c_\ell^* = c_\ell^p$, and if $I_\ell^* = I_\ell^u$ or I_ℓ^d then $c_\ell^* = 1$ denoting the full bandwidth allocation for offload/download. Finally, for a set of operation intervals $\mathcal{I}$, let $c(\mathcal{I}) = \sum_{I_\ell^* \in \mathcal{I}} c_\ell^*$.

In LHJS (Algorithm 1), we first divide $\mathcal{L}$ into two sets: the set of all light schedule instances $\mathcal{L}_L$ and the set of all heavy schedule instances $\mathcal{L}_H$. Next, we apply RandRound (Section IV-A) to obtain a solution for $\mathcal{L}_L$ and use SortSched (Section IV-B) to obtain a solution for $\mathcal{L}_H$. Finally, we select the solution with a higher energy saving as the final solution for EMJS. Here, we use vectors $\mathbf{y}$ and $\mathbf{z}$ to represent the optimal fractional solutions of LP problems LIS and HIS, respectively, to differentiate from the general solution $\mathbf{x}$. Notably, solving an LP problem with n variables optimally takes time $\mathcal{O}(n^3)$ [41], and is much faster than solving an ILP problem, which takes exponential time. Later in this section, we show that LHJS achieves a constant approximation ratio for EMJS.

Motivation for LHJS. Simultaneously considering schedule instances with computation resource allocations spanning 0 to 1 is very challenging to approximate. By scheduling $\mathcal{L}_L$ and $\mathcal{L}_H$ separately, we can obtain two additional properties that aid in deriving an approximation ratio for EMJS:

(i) When scheduling $\mathcal{L}_L$, if $\ell \in \mathcal{L}_L$ cannot be included in the solution due to conflict in I_ℓ^p, then at least $\frac{1}{2}$ of the resource of m_ℓ^p has been allocated to other schedule instances in the solution at some time $t \in I_\ell^p$.

(ii) When scheduling $\mathcal{L}_H$, since $c_\ell^p > \frac{1}{2}$ for all $\ell \in \mathcal{L}_H$, jobs cannot be scheduled in parallel on any server in the solution, i.e., two intervals $I_{\ell_1}^p$ and $I_{\ell_2}^p$ in the solution cannot overlap if $m_{\ell_1}^p = m_{\ell_2}^p$.

Algorithm 1 Light-Heavy Job Scheduling (LHJS)

Input: $\mathcal{N}$, $\mathcal{M}$
Output: $\mathcal{S}$

1: Define $\mathcal{L}$ by enumerating all possible schedule instances;
2: Divide $\mathcal{L}$ into $\mathcal{L}_L$ and $\mathcal{L}_H$;
3: Define an LP formulation (LIS) for $\mathcal{L}_L$ based on EMJS; let $\mathbf{y}$ be an optimal fractional solution of LIS.
4: $\mathcal{S}_L \leftarrow \texttt{RandRound}(\mathbf{y})$;
5: Define an LP formulation (HIS) for $\mathcal{L}_H$ based on EMJS; let $\mathbf{z}$ be an optimal fractional solution of HIS;
6: $\mathcal{S}_H \leftarrow \texttt{SortSched}(\mathbf{z})$;
7: **if** $e(\mathcal{S}_L) \geq e(\mathcal{S}_H)$ **do** $\mathcal{S} \leftarrow \mathcal{S}_L$;
else $\mathcal{S} \leftarrow \mathcal{S}_H$.

Algorithm 2 Randomized Rounding (RandRound)

Input: $\mathbf{y}$ (an optimal fractional solution of LIS)
Output: $\mathcal{S}_L$

1: $\mathcal{N}^{sel} \leftarrow \emptyset, \mathcal{L}^{sel} \leftarrow \emptyset, \mathcal{I}^{sel} \leftarrow \emptyset, \mathcal{S}_L \leftarrow \emptyset$;
/ Step 1: select job n_j with probability $y(\mathcal{L}_j)/\kappa$ */*
2: **for** $n_j \in \mathcal{N}$ **do**
3: $val_1 \leftarrow Random(0,1)$;
4: **if** $val_1 \leq y(\mathcal{L}_j)/\kappa$ **do** $\mathcal{N}^{sel} \leftarrow \mathcal{N}^{sel} \cup \{n_j\}$;
/ Step 2: select one ℓ with prob. $y_\ell/y(\mathcal{L}_j)$ for $n_j \in \mathcal{N}^{sel}$ */*
5: **for** $n_j \in \mathcal{N}^{sel}$ **do**
6: **for all** $\ell \in \mathcal{L}_j$ **do** $\tilde{y}_\ell \leftarrow y_\ell / y(\mathcal{L}_j)$;
7: $val_2 \leftarrow Random(0,1)$;
/ let ℓ^k be the k-th instance in $\mathcal{L}_j$ */*
8: **if** $\sum_{k=1}^{s-1} \tilde{y}_{\ell^k} < val_2 \leq \sum_{k=1}^{s} \tilde{y}_{\ell^k}$ **then**
9: $\mathcal{L}^{sel} \leftarrow \mathcal{L}^{sel} \cup \{\ell^s\}$;
/ Step 3: select operation intervals */*
10: **for** $m_i \in \mathcal{M}$ **do**
11: $\mathcal{I}_i^{sel} \leftarrow \emptyset$;
12: $\mathcal{I}_i \leftarrow \{I_\ell^* \mid \ell \in \mathcal{L}^{sel} \text{ and } \exists t, \mathbb{1}_i(t, I_\ell^*) = 1\}$;
13: Sort all $I_\ell^* \in \mathcal{I}_i$ in ascending order of $st(I_\ell^*)$;
14: **for** $I_\ell^* \in \mathcal{I}_i$ (from left to right) **do**
15: $\mathcal{I}_\ell^* \leftarrow \{I_{\ell'}^* \in \mathcal{I}_i^{sel} \mid \mathbb{1}_m(st(I_\ell^*), I_{\ell'}^*) = 1\}$;
16: **if** $c_\ell^* + c(\mathcal{I}_\ell^*) \leq 1$ **do** $\mathcal{I}_i^{sel} \leftarrow \mathcal{I}_i^{sel} \cup \{I_\ell^*\}$;
17: $\mathcal{I}^{sel} \leftarrow \mathcal{I}^{sel} \cup \mathcal{I}_i^{sel}$;
18: $\mathcal{S}_L \leftarrow \{\ell \in \mathcal{L}^{sel} \mid \forall I_\ell^* \in \ell, I_\ell^* \in \mathcal{I}^{sel}\}$.

A. The RandRound Algorithm for Light Schedule Instances

This subsection presents a randomized rounding algorithm, called RandRound, to obtain a feasible scheduling solution for $\mathcal{L}_L$. Based on the formulation of EMJS, we define a relaxed LP formulation corresponding to $\mathcal{L}_L$, called LIS, by relaxing x_ℓ into a continuous variable in the range $[0,1]$.

$$(\mathsf{LIS}) \quad \max \textstyle\sum_{\ell \in \mathcal{L}_L} e(\ell) \cdot x_\ell \tag{9}$$

subject to Eqs. (8a), (8b), (8c), and $x(\ell) \geq 0, \quad \forall \ell \in \mathcal{L}_L$. We denote the optimal fractional solution of LIS as $\mathbf{y}$. Let $y(\mathcal{L}_j) = \sum_{\ell \in \mathcal{L}_j} y_\ell$. Next, we use RandRound (Algorithm 2) to obtain a feasible scheduling solution for EMJS by rounding $\mathbf{y}$. Specifically, the function $Random(0,1)$ in lines 3 and 7 of RandRound samples a value of range $[0,1]$ *independently at random*, and the value of κ in line 4 of RandRound is determined in the proof of Theorem 1. Furthermore, we provide an example for operation interval selection (lines 11–16) in Fig. 2. In the following, we show that $\mathcal{S}_L$ obtained by RandRound is a feasible solution to EMJS, and derive the approximation ratio for RandRound.

Lemma 1. *The output $\mathcal{S}_L$ of* RandRound *is a feasible scheduling solution for* EMJS.

Proof: In lines 5–9, we choose at most one schedule instance for each job, so $\mathcal{S}_L$ satisfies constraint (8b) of EMJS. In line 16, we only add an operation interval into set $\mathcal{I}^{sel}$ when the resource capacity constraint is not violated. Thus, $\mathcal{S}_L$ also satisfies the constraints (8a) and (8b) of EMJS. ■

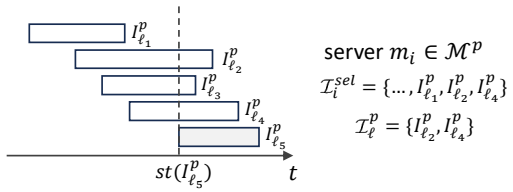


Fig. 2. An example of operation interval selection for server $m_i \in \mathcal{M}^p$, where we add $I^p_{\ell_5}$ into $\mathcal{I}_i^{sel}$ if $c^p_\ell + c(\mathcal{I}^p_\ell) \le 1$.

Theorem 1. *Suppose* $\mathbf{y}$ *is an optimal solution to* LIS*, and* $\mathcal{S}_L$ *is the schedule generated by* RandRound*. Let* $\kappa = 8$ *(line* 4 *of* RandRound*) and* $e(\mathbf{y}) = \sum_{\ell \in \mathcal{L}_L} e(\ell) \cdot y_\ell$*. Then, the expected value of* $e(\mathcal{S}_L)$ *satisfies the condition* $\mathbf{E}[e(\mathcal{S}_L)] \ge e(\mathbf{y})/16$.

Proof: Let $\ell \triangleq \langle m^u_\ell, m^p_\ell, m^d_\ell, I^u_\ell, I^p_\ell, I^d_\ell, c^p_\ell \rangle$ be a light schedule instance of job n_j. The probability of $\ell \in \mathcal{S}_L$,

$$\Pr[\ell \in \mathcal{S}_L] = \Pr[\ell \in \mathcal{L}^{sel}] \cdot \Pr[\ell \in \mathcal{S}_L \mid \ell \in \mathcal{L}^{sel}]. \quad (10)$$

Based on lines 3–9, we have $\Pr[n_j \in \mathcal{N}^{sel}] = y(\mathcal{L}_j)/\kappa$ and $\Pr[\ell \in \mathcal{L}^{sel} \mid n_j \in \mathcal{N}^{sel}] = y_\ell / y(\mathcal{L}_j)$. Thus, the first term on the right-hand side (RHS) of Eq. (10),

$$\Pr[\ell \in \mathcal{L}^{sel}] = \Pr[n_j \in \mathcal{N}^{sel}] \cdot \Pr[\ell \in \mathcal{L}^{sel} \mid n_j \in \mathcal{N}^{sel}] = \frac{y_\ell}{\kappa}. \quad (11)$$

For the second term on the RHS of Eq. (10),

$$\begin{aligned} \Pr[\ell \in \mathcal{S}_L \mid \ell \in \mathcal{L}^{sel}] &= \Pr[\forall I^*_\ell \in \ell, I^*_\ell \in \mathcal{I}^{sel} \mid \ell \in \mathcal{L}^{sel}] \\ &= 1 - \Pr[\exists I^*_\ell \in \ell, I^*_\ell \notin \mathcal{I}^{sel} \mid \ell \in \mathcal{L}^{sel}] \quad (12) \\ &\ge 1 - \textstyle\sum_{I^*_\ell \in \ell} \Pr[I^*_\ell \notin \mathcal{I}^{sel} \mid \ell \in \mathcal{L}^{sel}]. \end{aligned}$$

The inequality in Eq. (12) follows from the union bound. Then, we derive a bound on each term of the summation in Eq. (12).

We first analyze $\Pr[I^u_\ell \notin \mathcal{I}^{sel} \mid \ell \in \mathcal{L}^{sel}]$. When $I^*_\ell = I^u_\ell$, $c^*_\ell = 1$ and the offloading interval cannot overlap with each other at any vAP in a feasible scheduling solution. Thus,

$$\begin{aligned} &\Pr[I^u_\ell \notin \mathcal{I}^{sel} \mid \ell \in \mathcal{L}^{sel}] && && (13a)\\ =&\Pr[\exists I^u_{\ell'} \in \mathcal{I}^u_\ell, I^u_{\ell'} \in \mathcal{I}^{sel} \mid \ell \in \mathcal{L}^{sel}] && \text{(lines 15–16)} && (13b)\\ \le&\textstyle\sum_{I^u_{\ell'} \in \mathcal{I}^u_\ell} \Pr[I^u_{\ell'} \in \mathcal{I}^{sel} \mid \ell \in \mathcal{L}^{sel}] && \text{(union bound)} && (13c)\\ \le&\textstyle\sum_{I^u_{\ell'} \in \mathcal{I}^u_\ell} \Pr[\ell' \in \mathcal{L}^{sel} \mid \ell \in \mathcal{L}^{sel}] && \text{(line 12)} && (13d)\\ =&\textstyle\sum_{I^u_{\ell'} \in \mathcal{I}^u_\ell} \Pr[\ell' \in \mathcal{L}^{sel}] && \text{(line 3)} && (13e)\\ =&\textstyle\sum_{I^u_{\ell'} \in \mathcal{I}^u_\ell} \frac{y_{\ell'}}{\kappa} \le \frac{1}{\kappa} && \text{(Eqs. (11))} && (13f) \end{aligned}$$

The last inequality in Eq. (13f) holds due to Eq.(8a) and the fact that all $I^u_{\ell'} \in \mathcal{I}^u_\ell$ are active at time $st(I^u_\ell)$. Following a similar derivation, we have $\Pr[I^d_\ell \notin \mathcal{I}^{sel} \mid \ell \in \mathcal{L}^{sel}] \le \frac{1}{\kappa}$.

Next, we analyze $\Pr[I^p_\ell \notin \mathcal{I}^{sel} \mid \ell \in \mathcal{L}^{sel}]$. Since $\ell \in \mathcal{L}_L$, $0 < c^p_\ell \le \frac{1}{2}$. If I^p_ℓ cannot be added to $\mathcal{L}^{sel}$, we have $c(\mathcal{I}^p_\ell) \ge \frac{1}{2}$ at the time $st(I^p_\ell)$. Hence,

$$\Pr[I^p_\ell \notin \mathcal{I}^{sel} \mid \ell \in \mathcal{L}^{sel}] \le \Pr[c(\mathcal{I}^p_\ell) \ge \tfrac{1}{2} \mid \ell \in \mathcal{L}^{sel}]. \quad (14)$$

Next, we derive a bound on the RHS of Eq. (14). The expectation of $c(\mathcal{I}^p_\ell)$,

$$\mathbf{E}[c(\mathcal{I}^p_\ell) \mid \ell \in \mathcal{L}^{sel}] = \textstyle\sum_{I^p_{\ell'} \in \mathcal{I}^p_\ell} c^p_{\ell'} \cdot \Pr[I^p_{\ell'} \in \mathcal{I}^{sel} \mid \ell \in \mathcal{L}^{sel}].$$

Following a similar induction as Eqs.(13d) $\sim$ (13f) (the last inequality in Eq. (13f) will then hold due to Eq.(8b)), we have $\mathbf{E}[c(\mathcal{I}^p_\ell) \mid \ell \in \mathcal{L}^{sel}] \le \frac{1}{\kappa}$. Then, by applying Markov's inequality (i.e., $\Pr[X \ge a] \le \mathbf{E}[X]/a$), we have

$$\Pr[c(\mathcal{I}^p_\ell) \ge \tfrac{1}{2} \mid \ell \in \mathcal{L}^{sel}] \le \tfrac{1}{\kappa} \div \tfrac{1}{2} = \tfrac{2}{\kappa}.$$

Thus, by Eq. (14), we have $\Pr[I^p_\ell \notin \mathcal{I}^{sel} \mid \ell \in \mathcal{L}^{sel}] \le 2/\kappa$.

Substituting the results for I^u_ℓ, I^p_ℓ, and I^d_ℓ in Eq. (12), we have $\Pr[\ell \in \mathcal{S}_L \mid \ell \in \mathcal{L}^{sel}] \ge 1 - 4/\kappa$. Based on Eq. (10), $\Pr[\ell \in \mathcal{S}_L] \ge y_\ell \cdot (\frac{1}{\kappa} - \frac{4}{\kappa^2})$. When $\kappa = 8$, $(\frac{1}{\kappa} - \frac{4}{\kappa^2})$ reaches its maximum $1/16$. Thus, the expected value of $e(\mathcal{S}_L)$, $\mathbf{E}[e(\mathcal{S}_L)] = \sum_{\ell \in \mathcal{L}_L} \Pr[\ell \in \mathcal{S}_L] e(\ell) \ge \frac{1}{16} \sum_{\ell \in \mathcal{L}_L} e(\ell) y_\ell = e(\mathbf{y})/16$. ∎

B. The SortSched Algorithm for the Heavy schedule instances

In this subsection, we present an algorithm based on partial elimination ordering and a fractional local ratio method, referred to as SortSched, to obtain a feasible scheduling solution for $\mathcal{L}_H$. Based on the formulation of EMJS, we first define a relaxed LP formulation corresponding to $\mathcal{L}_H$, called HIS. When only considering $\mathcal{L}_H$, no two operation intervals related to the same machine $m \in \mathcal{M}$ can overlap at any time. Thus, HIS can be defined as follows.

$$(\mathsf{HIS}) \quad \max \textstyle\sum_{\ell \in \mathcal{L}_H} e(\ell) \cdot x_\ell \qquad \text{subject to:} \quad (15)$$

$$\textstyle\sum_{\ell \in \mathcal{L}_H, I^*_\ell \in \ell, \mathbb{1}_i(t, I^*_\ell) = 1} x_\ell \le 1, \forall 1 \le t \le \Delta, \forall m_i \in \mathcal{M}, \quad (16)$$

Eq. (8c), and $x(\ell) \ge 0, \forall \ell \in \mathcal{L}_H$. For a heavy schedule instance ℓ of job n_j, let $\mathcal{L}(\ell)$ be the set of heavy schedule instances belonging to the same job n_j, and $\mathcal{A}(\ell)$ be the set of heavy schedule instances whose operation intervals (at least one) overlap with ℓ but do not belong to $\mathcal{L}(\ell)$. We first show the following property for any feasible solution of HIS.

Proposition 1. *Let* $\mathbf{x}$ *be any feasible solution to* HIS*. There is a schedule instance* ℓ *satisfying* $x_\ell + \sum_{\ell' \in \mathcal{A}(\ell)} x_{\ell'} \le 6$.

Proof: For two intersecting schedule instances ℓ and ℓ' (i.e., $\ell' \in \mathcal{A}(\ell)$ and $\ell \in \mathcal{A}(\ell')$), let $q(\ell, \ell') = x_\ell \cdot x_{\ell'}$. Besides, $q(\ell, \ell) = (x_\ell)^2$. For an operation interval $I^*_\ell \in \ell$, let $\mathcal{R}(I^*_\ell)$ be the set of schedule instances in $\mathcal{L}_H$ that have an operation interval intersecting the right end of I^*_ℓ (including ℓ itself). Consider $\sum_{\ell \in \mathcal{L}_H} (q(\ell, \ell) + \sum_{\ell' \in \mathcal{A}(\ell)} q(\ell, \ell'))$. We first obtain an upper bound of this sum as follows.

For each operation interval I^*_ℓ of ℓ, we sum up $q(\ell, \ell')$ for all schedule instances ℓ' having at least one operation interval that intersects with I^*_ℓ (including ℓ itself). We argue that *it suffices to sum up* $q(\ell, \ell')$ *only for schedule instances* $\ell' \in \mathcal{R}(I^*_\ell)$ *and then multiply the total sum by* 2. This is because if an operation interval $I^*_{\ell'}$ of schedule instance ℓ' intersects with I^*_ℓ, we have either $\ell' \in \mathcal{R}(I^*_\ell)$ or $\ell \in \mathcal{R}(I^*_{\ell'})$. Since $q(\ell, \ell') = q(\ell', \ell)$,

$$\sum_{\ell \in \mathcal{L}_H} (q(\ell, \ell) + \sum_{\ell' \in \mathcal{A}(\ell)} q(\ell, \ell')) \le 2 \sum_{\ell \in \mathcal{L}_H} \sum_{I^*_\ell \in \ell} \sum_{\ell' \in \mathcal{R}(I^*_\ell)} q(\ell, \ell'). \quad (17)$$

Based on constraint (16) of HIS and the definition of $q(\ell, \ell')$,

$$\textstyle\sum_{\ell' \in \mathcal{R}(I^*_\ell)} q(\ell, \ell') \le x_\ell \cdot \sum_{\ell' \in \mathcal{R}(I^*_\ell)} x_{\ell'} \le x_\ell. \quad (18)$$

Algorithm 3 Sort Scheduling (`SortSched`)

Input: $\mathbf{z}$ (an optimal fractional solution of HIS)
Output: $\mathcal{S}_H$

1: $\mathcal{F} \leftarrow \emptyset$. Remove all ℓ with $z_\ell = 0$ from $\mathcal{L}_H$;
2: **while** $\mathcal{L} \neq \emptyset$ **do**
3: Find a schedule instance ℓ that satisfies the inequality of Proposition 1;
4: Append ℓ to the end of $\mathcal{F}$, i.e., $\mathcal{F} \leftarrow \mathcal{F} \cup \{\ell\}$;
5: Remove ℓ from $\mathcal{L}_H$, and set $z_\ell = 0$;
6: $\mathcal{S}_H \leftarrow$ `FracLR`($\mathcal{F}$, $\mathbf{e}$);

Algorithm 4 Fractional Local Ratio (`FracLR`)

`FracLR`($\mathcal{U}$, $\mathbf{w}$):

1: Remove all ℓ with non-positive energy $w(\ell)$ from $\mathcal{U}$.
2: **if** $\mathcal{U} = \emptyset$, **return** $\emptyset$.
3: Choose the ℓ with smallest index from $\mathcal{U}$.
4: Decompose the energy vector $\mathbf{w} = \mathbf{w_1} + \mathbf{w_2}$ such that $w_1(\ell') = w(\ell)$ if $\ell' \in \mathcal{L}(\ell) \cup \mathcal{A}(\ell)$; otherwise, $w_1(\ell') = 0$.
5: $\mathcal{S}'_F \leftarrow$ `FracLR`($\mathcal{U}$, $\mathbf{w_2}$).
6: **if** $\mathcal{S}'_F \cup \{\ell\}$ is a feasible schedule, **return** $\mathcal{S}_F \triangleq \mathcal{S}'_F \cup \{\ell\}$; otherwise, **return** $\mathcal{S}_F \triangleq \mathcal{S}'_F$.

Using Eqs. (17) and (18), and the fact that each schedule instance has 3 operation intervals, we can get

$$\sum_{\ell \in \mathcal{L}_H} (q(\ell,\ell) + \sum_{\ell' \in \mathcal{A}(\ell)} q(\ell,\ell')) \leq 2 \sum_{\ell \in \mathcal{L}_H} \sum_{I^*_\ell \in \ell} x_\ell \leq 6 \sum_{\ell \in \mathcal{L}_H} x_\ell. \quad (19)$$

Therefore, we can conclude that there exists at least one (otherwise Eq. (19) will not hold) schedule instance ℓ satisfies

$$q(\ell,\ell) + \sum_{\ell' \in \mathcal{A}(\ell)} q(\ell,\ell') = (x_\ell)^2 + \sum_{\ell' \in \mathcal{A}(\ell)} x_{\ell'} \cdot x_\ell \leq 6 \cdot x_\ell. \quad (20)$$

This lemma is proved by factoring out x_ℓ from both sides of the inequality of Eq. (20). ■

Let $\mathbf{z}$ be an optimal fractional solution to the LP problem HIS. Next, we apply `SortSched` (Algorithm 3) to obtain a scheduling solution with respect to $\mathcal{L}_H$. Since $\mathbf{z}$ is a feasible solution, Proposition 1 applies. Thus, we first sort all schedule instances ℓ with positive z_ℓ as shown in lines 2–5 of `SortSched`. Note that we append the new schedule instance to the end of $\mathcal{F}$ for every while loop (line 4), so the resulting set $\mathcal{F}$ is already sorted. Next, we show that we can always find such a schedule instance ℓ in line 3 of `SortSched`.

Lemma 2. *Let $\mathcal{F}$ be the resulting set of schedule instances from lines 2–5 of* `SortSched`*, and let ℓ^k denote the k-th schedule instance in $\mathcal{F}$. Let $\mathcal{F}[k] = \{\ell^k, \ell^{k+1}, ..., \ell^{|\mathcal{F}|}\}$. Then, we have $z_{\ell^k} + \sum_{\ell' \in \mathcal{A}(\ell^k) \cap \mathcal{F}[k]} z_{\ell'} \leq 6$.*

Proof: By removing z_{ℓ^s} from $\mathbf{z}$ for $s < k$, the resulting $\mathbf{z}$ is still a feasible solution to HIS. Thus, Proposition 1 still applies to ℓ^k by only considering its neighbors in $\mathcal{F}[k]$. ■

After we obtained the set of sorted schedule instance $\mathcal{F}$, we apply `FracLR` (Algorithm 4) to $\mathcal{F}$ to obtain a feasible scheduling solution for EMJS (line 6 of `SortSched`), where the $\mathbf{e}$ is the saved energy vector of all schedule instances. `FracLR` is a recursive algorithm, and in each recursive layer of `FracLR`, we *decompose the saved energy of schedule instances such that each schedule instance selection maintains a constant local approximation ratio corresponding to the optimal fractional solution* $\mathbf{z}$ *of* HIS. Because the saved energy of each schedule instance is updated (decomposed) in each recursive layer, we use $w(\ell)$ to represent the updated saved energy of schedule instance ℓ in each layer ($e(\ell)$ denotes the original saved energy of ℓ). The initial call to this recursive algorithm is `FracLR`($\mathcal{F}$, $\mathbf{e}$). Next, we show that `SortSched` is a 7-approximation algorithm for HIS.

Theorem 2. *Suppose $\mathbf{z}$ is an optimal solution to* HIS*, and $\mathcal{S}_H$ is the schedule returned by* `FracLR` *in line 6 of* `SortSched`*. Then, it holds that $e(\mathcal{S}_H) \geq \frac{1}{7} \cdot \mathbf{e} \cdot \mathbf{z}$.*

Proof: Let $w(\mathcal{S}_F) = \sum_{\ell \in \mathcal{S}_F} w(\ell)$. Note that any schedule instance removed by `FracLR` in step 1 is considered to have zero weight. For ease of understanding, we denote the innermost recursive layer (when set $\mathcal{U} = \emptyset$) as layer 0, and the outermost layer (initial call to `FracLR`) as layer U. Besides, we use superscript i to denote the parameters corresponding to recursive layer i, i.e., $\mathbf{w}^i$ denotes the energy vector $\mathbf{w}$ corresponding to recursive layer i. We prove this lemma by showing that $w^i(\mathcal{S}^i_F) \geq \frac{1}{7} \cdot \mathbf{w}^i \cdot \mathbf{z}$ for all $i = 0, 1, ..., U$.

In the base case ($i = 0$), $\mathcal{S}^0_F = \emptyset$ and the inductive hypothesis holds, since the weight vector $\mathbf{w}^0$ is considered to be zero. Next, we prove the inductive step.

For $i \geq 1$, suppose $w^{i-1}(\mathcal{S}^{i-1}_F) \geq \frac{1}{7} \cdot \mathbf{w}^{i-1} \cdot \mathbf{z}$. According to line 4 of `FracLR`, $\mathbf{w}^i_\mathbf{2}$ is equivalent to $\mathbf{w}^{i-1}$ *before* all schedule instances with non-positive energy are removed from $\mathcal{U}^{i-1}$. Adding non-positive component back to $\mathbf{w}^{i-1}$ will only decrease the value of $\mathbf{w}^i_\mathbf{2} \cdot \mathbf{z}$, thus, $w^{i-1}(\mathcal{S}^{i-1}_F) \geq \frac{1}{7} \cdot \mathbf{w}^{i-1} \cdot \mathbf{z} \geq \frac{1}{7} \cdot \mathbf{w}^i_\mathbf{2} \cdot \mathbf{z}$. Besides, in layer $i-1$, we have removed all schedule instances ℓ with nonpositive energy $w^{i-1}(\ell)$, and they will never be added to $\mathcal{S}^{i-1}_F$, thus, we have

$$w^i_2(\mathcal{S}'^i_F) = w^i_2(\mathcal{S}^{i-1}_F) = w^{i-1}(\mathcal{S}^{i-1}_F).$$

Since $w^i_1(\ell^i) = w^i(\ell^i)$, $w^i_2(\ell^i) = 0$; thus, $w^i_2(\mathcal{S}^i_F) = w^i_2(\mathcal{S}'^i_F)$. Hence, $w^i_2(\mathcal{S}^i_F) = w^{i-1}(\mathcal{S}^{i-1}_F) \geq \frac{1}{7} \cdot \mathbf{w}^i_\mathbf{2} \cdot \mathbf{z}$.

In i-th recursive layer of `FracLR`, the chosen schedule instance ℓ^i has the smallest index in $\mathcal{U}^i$. According to Lemma 2 and step 3 of `FracLR`, $\sum_{\ell' \in \mathcal{A}(\ell) \cap \mathcal{U}^i} z_{\ell'} \leq 6$. Due to constraint (8c), we have $\sum_{\ell' \in \mathcal{L}(\ell)} z_{\ell'} \leq 1$. Thus, $\mathbf{w}^i_\mathbf{1} \cdot \mathbf{z} \leq 7 w^i(\ell)$. For the returned solution $\mathcal{S}_F$, it either contains schedule instance ℓ or contains at least one schedule instance in $\mathcal{L}(\ell) \cup \mathcal{A}(\ell)$. In both cases, $w^i_1(\mathcal{S}_F)$ is at least $w^i(\ell)$. Thus, $w^i_1(\mathcal{S}^i_F) \geq \frac{1}{7} \mathbf{w}^i_\mathbf{1} \cdot \mathbf{z}$.

Since the energy vector is decomposed in each layer such that $\mathbf{w}^i = \mathbf{w}^i_\mathbf{1} + \mathbf{w}^i_\mathbf{2}$, and the objective function is a linear multiplication of $\mathbf{w}$ and $\mathbf{z}$, we have

$$w^i(\mathcal{S}^i_F) = w^i_1(\mathcal{S}^i_F) + w^i_2(\mathcal{S}^i_F) \geq \tfrac{1}{7} \mathbf{w}^i_\mathbf{1} \cdot \mathbf{z} + \tfrac{1}{7} \mathbf{w}^i_\mathbf{2} \cdot \mathbf{z} = \mathbf{w}^i \cdot \mathbf{z}.$$

Based on the simple induction, we have $w^i(\mathcal{S}^i_F) \geq \frac{1}{7} \cdot \mathbf{w}^i \cdot \mathbf{z}$ for all $i = 0, 1, ..., U$. Since the initial call to `FracLR` (when $i = U$) is `FracLR`($\mathcal{F}$, $\mathbf{e}$), the theorem is proved. ■

Theorem 3. *Let $\mathcal{S}$ be the scheduling solution obtained by* LHJS*, and $\mathcal{S}^*$ be the optimal scheduling solution to* EMJS*. Then, the expected value of $e(\mathcal{S})$ satisfies $\mathbf{E}[e(\mathcal{S})] \geq \frac{1}{23}e(\mathcal{S}^*)$.*

Proof: Let OPT_L denote the optimal objective value of LIS, and OPT_H denote the optimal objective value of HIS. According to Theorem 1 and Theorem 2, we have $OPT_L \leq 16\mathbf{E}[e(\mathcal{S}_L)]$ and $OPT_H \leq 7e(\mathcal{S}_H)$. Furthermore, based on line 7 of LHJS, we have $e(\mathcal{S}) \geq \max\{e(\mathcal{S}_L), e(\mathcal{S}_H)\}$. Since any feasible solution to EMJS can be divided into a set of light schedule instances and a set of heavy schedule instances, we have $e(\mathcal{S}^*) \leq OPT_L + OPT_H \leq 16\mathbf{E}[e(\mathcal{S}_L)] + 7e(\mathcal{S}_H) \leq 23\mathbf{E}[e(\mathcal{S})]$. Therefore, $\mathbf{E}[e(\mathcal{S})] \geq \frac{1}{23}e(\mathcal{S}^*)$. ∎

Discussion. The partial elimination ordering technique employed in SortSched originates from Bar-Yehuda et al. [42], initially devised for scheduling jobs given multiple processing intervals on a single machine. Building upon their approach, we tackle the more complex sub-problem (HIS): scheduling jobs with multiple operations, with multiple unrelated candidates (machines) for each operation. Because the total number of schedule instances depends on the total number of time units Δ, LHJS is a pseudo-polynomial approximation algorithm for offline EMJS. The offline scheduling algorithm can be used in applications where workloads are known apriori, such as intelligent patrol robots within factories or forest inspection drones that have regular working schedules, moving paths, and workloads (e.g., anomaly detection).

V. Online Scheduling Algorithm for Online EMJS

In this section, we consider online EMJS, where the scheduler only knows a job's information after its release in the system. In the online scenario, the scheduler is deployed on a central server within the backhaul network, while other servers and vAPs continuously update their resource utilization to the scheduler via the wired backhaul network. When jobs are generated, their meta-information is offloaded to the scheduler through accessible vAPs. Given that the size of job meta-information is typically very small, we omit its transmission time. The scheduler is triggered whenever a job arrives. To address online EMJS, we introduce the Load Balanced Job Scheduling Algorithm (LBS) invoked whenever a job is released in the system. The intuition behind LBS is to minimize the impact of the current job's scheduling on future job arrivals, such that more energy can be saved for future jobs. LBS is outlined in Algorithm 5.

Let $\mathcal{N}(t_{cur})$ be the set of jobs whose meta-information arrives at the online scheduler at time t_{cur}. Let $\omega = \langle n_j, T_u, T_d\rangle$ be a ring coverage window selection for job n_j, where $T_u, T_d \in \psi_j$ are the selected offloading and downloading ring coverage windows, respectively. Since the energy saving of job n_j, e_j^{save}, depends on the selection of ring coverage windows, we first enumerate all possible window selection candidates ω for all jobs $n_j \in \mathcal{N}(t_{cur})$ and compute the corresponding saved energy, denoted as $e(\omega)$ (lines 2–5).

Next, we consider the candidate $\omega = \langle n_j, T_{ur}, T_{ds}\rangle$ with the largest saved energy. We first determine the earliest starting time for offloading, t_j^u, and the latest starting time for downloading, t_j^d (lines 8–11). For each candidate server $m_i \in \mathcal{M}_j^p$, we determine job n_j's feasible processing interval $[t_e, t_l]$, and then determine the smallest resource allocation option c_{ij} such that n_j can be processed within $[t_e, t_l]$ without violating server resource capacity constraint. If c_{ij} can be found, we compute the resource usage (line 17) of server m_i during $[t_e, t_l]$ if job n_j is processed on server m with resource allocation c_{ij} (lines 13–19). After all candidate servers have been checked, we choose the server with the least resource usage as the processing server m_p for job n_j. Then, we determine the earliest processing starting time, t_j^p, while minimizing the server's peak resource usage (line 22). Finally, based on t_j^p, we update t_j^d to the earliest downloading starting time. The algorithm stops when the candidate set, Ω, becomes empty.

Algorithm 5 Load Balanced Job Scheduling (LBS)

Input: $\mathcal{N}(t_{cur})$, $\mathcal{M}$
Output: $\mathcal{S}$

1: Initialize $\mathcal{S} \leftarrow \emptyset, \Omega \leftarrow \emptyset$, and $\mathcal{N}^{sel} \leftarrow \emptyset$;
2: **for all** $n_j \in \mathcal{N}(t_{cur}), T_u \in \psi_j, T_d \in \psi_j$ **do**
3: $\quad\omega \leftarrow \langle n_j, T_u, T_d\rangle$;
4: $\quad$**if** $\exists t_j^u, t_j^d$ satisfying Eqs. (2), (3), (6), and (7) **then**
5: $\quad\quad$compute $e(\omega)$ as in Subsection III-B;
6: $\quad\quad$Add ω into Ω if $e(\omega) > 0$;
7: sort Ω in non-increasing order of $e(\omega)$;
8: **while** $\Omega \neq \emptyset$ **do**
9: $\quad\mathcal{U} \leftarrow \emptyset, \Omega \leftarrow \Omega \setminus \{\omega\}$;
10: $\quad$Let $\omega = \langle n_j, T_{ur}, T_{ds}\rangle$ be the leftmost candidate in Ω;
11: $\quad$**if** $n_j \in \mathcal{N}^{sel}$ **then** go to line 7;
12: $\quad$Let t_j^u be the earliest time such that vAP m_u is idle during $[t_j^u, t_j^u + d_j^u - 1]$ and t_j^u satisfies Eqs. (2), (3);
13: $\quad$Let t_j^d be the latest time such that vAP m_d is idle during $[t_j^d, t_j^d + d_j^d - 1]$ and t_j^d satisfies Eqs. (6) and (7);
14: $\quad$**for** $m_i \in \mathcal{M}_j^p$ **do**
15: $\quad\quad$Determine the earliest and latest times feasible for processing t_e and t_l based on t_j^u, t_j^d, Eqs. (4), (5);
$\quad\quad$*/* let $\beta_i[t]$ be the used resource fraction of m_i at time t */*
16: $\quad\quad$**for** $c \in \mathcal{C}_i$ (from smallest to largest) **do**
17: $\quad\quad\quad$**if** $\exists t'$ satisfying $\beta_i[t] + c \leq 1, \forall t \in [t', t' + d_j^p - 1]$, Eqs. (4) and (5) **then**
18: $\quad\quad\quad\quad U_i = (c \cdot d_j^p + \sum_{t=t_e}^{t_l} \beta_i[t])/(t_l - t_e + 1)$;
19: $\quad\quad\quad\quad\mathcal{U} \leftarrow \mathcal{U} \cup \{U_i\}, c_{ij} \leftarrow c$;
20: $\quad\quad\quad\quad$**break**;
21: $\quad$**if** $\mathcal{U} = \emptyset$ **then** go to line 7;
22: $\quad m_p \leftarrow \arg\min_{m_i \in \mathcal{M}_j^p} U_i$;
23: $\quad$Given the resource allocation c_{pj} (line 18), let t_j^p be the earliest time meeting the condition in line 16 and has smallest $\max\{\beta_p[t] + c_{pj} \mid t \in [t_j^p, t_j^p + d_j^p - 1]\}$;
24: $\quad$Update t_j^d to the earliest time such that vAP m_{ds} is idle during $[t_j^d, t_j^d + d_j^d - 1]$ and t_j^d satisfies Eqs. (5)∼(7);
25: $\quad$Update resource allocation status for all $m_i \in \mathcal{M}$;
26: $\quad\mathcal{S} \leftarrow \mathcal{S} \cup \{\langle m_u, m_p, m_d, t_j^u, t_j^p, t_j^d, c_{pj}\rangle\}, \mathcal{N}^{sel} \leftarrow \mathcal{N}^{sel} \cup \{n_j\}$;

VI. Experimental Evaluation

This section evaluates the performances of LHJS and LBS. We first assess LHJS for offline job scheduling by comparing it with an offline algorithm in the literature. Then, we evaluate LBS for online job scheduling against its (three) variants. The experiments were conducted on a desktop PC equipped with an Intel(R) Xeon(R) W-2235 3.8GHz CPU and 32G RAM[1].

A. Simulation Setup

We select a city area of $1200m \times 700m$ using OpenStreetMap [43]. Traffic data are generated using SUMO [44], simulating 675 vehicles randomly entering and leaving this area within 600 seconds. Each AP has a height of $30m$, and its network is divided into 2 rings with coverage ranges of $0 \sim 100m$ and $100 \sim 200m$. We consider a MEC with 13 APs, 6 GPU servers, and 6 CPU servers (M^u=M^d=13, M^p=12). The geographic coordinates of APs are determined with CellMapper [45], and servers are co-located with APs.

Nvidia Jetson Nano [46] is used as ED, and the wireless network data is obtained by profiling the wireless communication between Jetson Nano and a WiFi-5 router (TL-WDR8620). For each $m_i \in \mathcal{M}^u$, α_i is set to 40 or 80 MHz. When $\alpha_i = 40$ (likewise 80) MHz, the data offloading rate η_j^u is set to 33 and 23 (likewise 66 and 46.5) MBps for ring 1 and 2, respectively. For each vAP $m_i \in \mathcal{M}^d$, the data downloading rate η_j^d is set to 38 and 77 MBps for $\alpha_i = 40$ and $\alpha_i = 80$ MHz, respectively. We consider 3 types of CPU (Intel w2235, 10700k and 14700k, with computing units $12, 16$, and 16, respectively) and 3 types of GPU (Nvidia 2080Ti, 4060Ti and 3090 with computing units $8, 8$, and 8). Notably, with the same number of computing units, the processing time for the same job can differ with different server hardware.

We consider 3 GPU applications (resnet101, resnet152, vgg-16) for object detection and 1 CPU application (surf3D) for 3D-object surface reconstruction. The input θ_j^{in} of GPU applications includes 38 images ranging from 0.01 to 1.2 MB, and that of surf3D includes 8 3D mesh objects ranging from 0.14 to 0.6 MB. For jobs of application resnet101, resnet152, vgg-16 and surf3D, their release times γ_j are sampled from the ranges $[1, 100]$, $[1, 70]$, $[1, 90]$, and $[1, 50]$ milliseconds (ms), respectively, and their deadlines δ_j are set to 80, 110, 90, and 130 ms, respectively. The job (local and remote) processing time and ED's energy consumption for local processing, offloading, and downloading are obtained through profiling. Specifically, the power for processing CPU applications locally ranges from 0.97 to 1.11 Watts, and that for GPU applications ranges from 1.8 to 5.33 Watts. Besides, the average power of Jetson Nano for data offloading and downloading is 2.08 and 2.13 Watts, respectively.

In this experiment, we evaluate the algorithms' performance on jobsets with varying resource utilizations. The job deadlines are approximately 100 ms; therefore, we use a scheduling window of 180 ms for each jobset, since using a larger scheduling window does not necessarily lead to increased resource utilization of the jobset. The window starting times are randomly sampled from 120 to 300 seconds from the 600 second simulation. We only consider the simulation period where most of the vehicles are active. Jobs are then randomly mapped to active vehicles during the scheduling window, obtaining ring coverage windows ψ_j for each job by tracing the physical position of vehicles. Given a job mapping and computation resource allocation c_j for job n_j, the computation resource utilization of job n_j in a MEC is defined as $U_j^c = c_j \cdot d_j^p / (D \cdot d_j^{p,max})$, where $d_j^{p,max}$ is the maximum allowable processing duration for job n_j under the given mapping, and D is the number of servers in the MEC. Considering various job mappings and c_j, let $U_j^{c,min}$ be the smallest utilization among all U_j^c of job n_j. The total *computation resource utilization* of a jobset is defined as $u^c = \sum_{j \in \mathcal{N}} U_j^{c,min}$, and the total uplink bandwidth utilization u^b of a jobset is analogously defined. To generate jobsets with various computation and bandwidth resource utilization, we consider jobset sizes J from a range of $[60, 160]$ (in increments of 10). In total, we generate 3000 jobsets with various utilizations.

Offline Baseline. For offline experiments, we compare LHJS with a heuristic algorithm (SortAll) derived from SortSched and an approximation algorithm (SEARCH) proposed by Zhu *et al.* [18]. The key difference between SortAll and SortSched is that SortAll considers all schedule instances, rather than only heavy schedule instances. SEARCH first sorts all jobs in ascending order of deadlines and divides every consecutive q jobs into a group; then, it applies an exhaustive search within each group to find the optimal schedule. Unlike our study, Zhu *et al.* [18] did not consider (i) job waiting time for processing/downloading, (ii) varying computation resource allocation, or (iii) job mapping for processing/downloading. Therefore, SEARCH cannot be trivially extended to EMJS. However, the exhaustive search concept in SEARCH is applicable to any scheduling problem, including ours, making it a relevant baseline. To adapt SEARCH to EMJS, we search for every possible job mapping and operation ordering within each group and set the group size $q = 30$ to ensure that SEARCH has a runtime comparable to LHJS. Notably, neither SortAll nor SEARCH provide an approximation guarantee for EMJS.

Online Baseline. For online experiments, we compare LBS with its variants: LBSLate, LCEarly, and LCLate. Compared with LBS, LBSLate schedules each job's operation as late as possible, LCEarly considers only the largest computation resource allocation option for job processing, and LCLate considers only the largest computation resource allocation option and schedules each job's operation as late as possible.

Metric. We use the *Performance Ratio* R to measure the performance of all algorithms, where R is defined as the ratio of the total saved energy by an algorithm to the optimal saved energy for EMJS. Obtaining the optimal saved energy for EMJS is challenging; hence, we utilize the optimal solution of a relaxed LP formulation of EMJS, computed with the LP solver Gurobi (with a timeout limit of 10 minutes), as an upper

[1]Experiments code is available at https://github.com/CPS-research-group/CPS-NTU-Public/tree/RTSS2024.

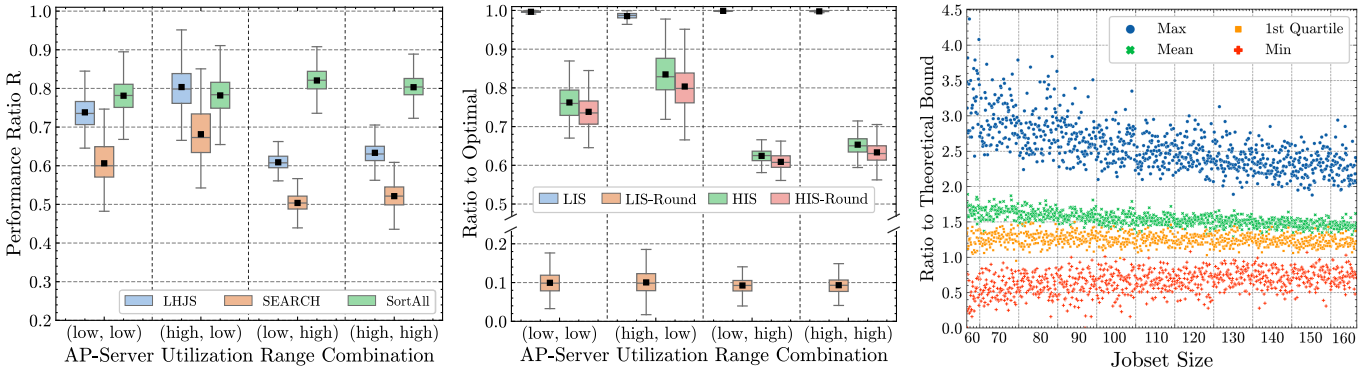


Fig. 3. (a) Performance ratios of LHJS, SEARCH, and SortAll under different AP-server utilization range combinations in offline job scheduling; (b) Intermediate results of LHJS compared to the optimal saved energy under different AP-server utilization range combinations. LIS-Round (likewise HIS-Round) is the integral solution for all light (likewise heavy) schedule instances via RandRound (likewise SortSched); (c) A comparison between the integral solution obtained by RandRound and the approximation bound of RandRound (as in Theorem 1).

bound of the optimal to compute R. Hence, the computed R is in fact a lower bound of the algorithm's actual performance.

B. Offline Scheduling Experiment Result Discussion

LHJS *Performance.* Algorithms' performance are illustrated under different combinations of AP-Server utilization (i.e., (u^b, u^c)), where two different utilization ranges are employed for both APs and servers: **low** ($[0, 0.9)$) and **high** ($[1.1, \inf)$). The performances of LHJS SEARCH, and SortAll under various utilization range combinations are shown in Fig. 3(a), while insights into the intermediate results of LHJS are provided in Fig. 3(b). On average, LHJS, SEARCH, and SortAll achieve performance ratios of 69.8%, 58%, and 79.5%, respectively. It is evident that LHJS consistently outperforms SEARCH across all utilization range combinations. Additionally, SortAll demonstrates superior performance when the computation resource utilization is high. While the performance of LHJS and SEARCH shows a monotonically increasing trend with AP utilization and a decreasing trend with server utilization, SortAll maintains stable performance across different utilization range combinations.

As shown in Fig. 3(b), although the optimal fractional solution of LIS exceeds that of HIS, the rounding loss induced by RandRound is significantly greater than SortSched. Consequently, a larger integral solution is obtained for $\mathcal{L}_H$ compared to $\mathcal{L}_L$. Thus, the performance of LHJS is heavily influenced by the performance of SortSched. Furthermore, since the rounding loss of SortSched is relatively small (around 2.3%), the practical performance of LHJS is primarily constrained by the optimal solution of HIS. When the server utilization is low, the increasing AP utilization reduces the allowable time for job processing, favouring a large resource allocation option to shorten processing duration. Therefore, the optimal solution of HIS closely aligns with the optimal solution of EMJS, leading to enhanced performance of LHJS. When the server utilization increases from low to high, server resources become more scarce, and always allocating full server resources to a job may lead to a shortage of computing resources, resulting in a decreased optimal solution for HIS. It is important to note that the objective value of HIS serves as an upper bound for all scheduling algorithms that assume full computation resource allocation. Considering the rounding loss of SortSched (2.3%), SortSched offers a near-optimal practical solution for scenarios where fractional computation resource allocation is not permitted.

Based on Figs. 3(a) and 3(b), we observe that the practical performance of LHJS is limited by only considering solutions from either light or heavy schedule instances to achieve a constant approximation ratio. In contrast, SortAll considers all schedule instances without aiming for theoretical guarantees, resulting in better practical performance compared to LHJS. This also highlights that approximation algorithms can serve as a valuable foundation for designing heuristic algorithms with good practical performance.

RandRound *Evaluation.* As RandRound is a randomized algorithm, we also evaluate its actual rounding performance. We conduct experiments with 900 jobsets featuring various resource utilization combinations and sizes. For each jobset, utilizing the computed optimal fractional solution of LIS, we execute RandRound 50 times and record the resulting integral solutions. Among these 50 integral solutions, we select 4 results (maximum, minimum, average, and 1st quartile) and illustrate the ratios of these results to the approximation bound (Theorem 1) in Fig. 3(c). A higher ratio indicates better practical performance of RandRound. The 1st quartile denotes that 25% of the 50 integral solutions possess saved energies lower than the 1st quartile. Observably, the ratio corresponding to the 1st quartile of nearly all jobsets surpasses 1, indicating a 75% probability that the integral solution obtained by RandRound exceeds its theoretical bound. Additionally, the variance of the random rounding diminishes with increasing jobset size.

LHJS *Scalability.* To evaluate the scalability of LHJS, we consider two additional MECs with larger scales: *medium-scale* MEC (with 16 APs and 18 servers) and *large-scale*

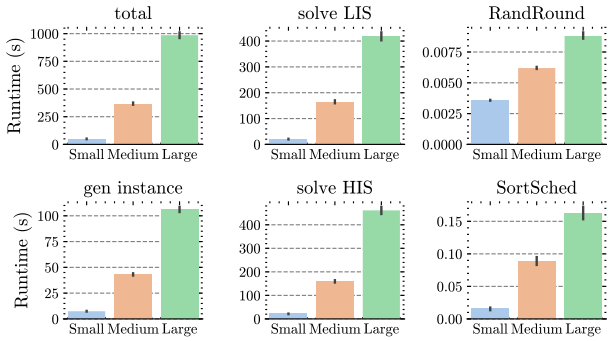


Fig. 4. Runtime analysis of LHJS for different scales of MEC

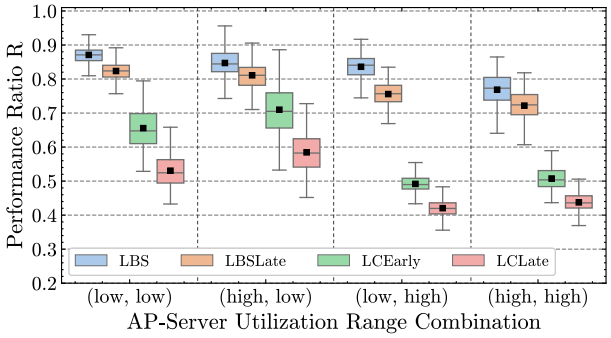


Fig. 5. Performance ratios of LBS and its variants under different AP-server utilization range combinations for online job scheduling

MEC (with 21 APs and 24 servers). For each scale of MEC (including the base MEC which we now denote as *small-scale* MEC), we generate 100 jobsets with total resource utilizations u^c and u^b ranging from 0.9 to 1.1. Specifically, for small, medium, and large-scale MECs, each generated jobset contains 80, 140, and 200 jobs, respectively. In Fig. 4, we present the average runtimes of LHJS for different scales of MECs. The subfigure "total" represents the overall runtime of LHJS, while the other five subfigures depict the runtime of individual steps of LHJS (i.e., "gen instance" denotes the time taken to define all schedule instances). It's evident that more than 10% of the runtime is devoted to defining schedule instances, while over 85% of the time is spent on solving the LP formulations LIS and HIS. The time spent on LP rounding (RandRound and SortSched) accounts for less than 0.1% of the total runtime. Furthermore, the runtime of LHJS notably increases with the growing number of APs and servers in the MEC. This increase is primarily attributed to the significant rise in the number of schedule instances with the expansion of system scales and jobset sizes, posing a considerable challenge for the LP solver.

C. Online Scheduling Experiment Result Discussion

LBS *Performance.* In Fig. 5, we present the performance of LBS and its variants under different AP-Server utilization range combinations for the base MEC (small-scale). On average, LBS achieves a performance ratio of 83.2% compared to the offline optimal solution of EMJS, surpassing its variants LBSLate, LCEarly, and LCLate by 5.2%, 23.7%, and 33.7% respectively. Notably, the result highlights the significance of the computation resource allocation strategy in online scheduling compared to the policy for scheduling each job's operation (i.e., earliest or latest). Additionally, the performance gap between LBS and LCEarly widens as server utilization increases, which may stem from inefficient resource utilization with a large resource allocation option and the potentially significant impact of such an allocation on future job arrivals.

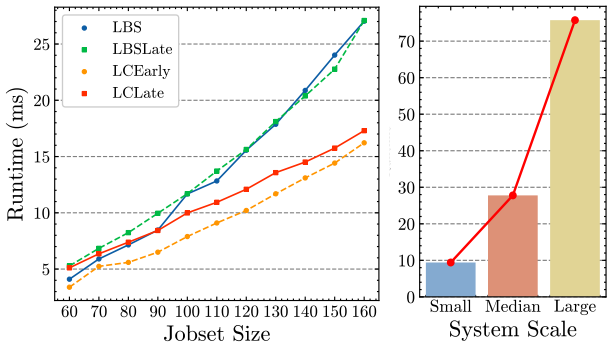


Fig. 6. Runtime of LBS under different jobset sizes and MEC scales

LBS *Scalability.* Fig. 6 illustrates the average runtime of LBS across different jobset sizes (in small-scale MEC) and varying MEC scales. It is evident that the LBS runtime increases almost linearly with the increase of jobset size. Despite considering various resource allocation options in LBS, its runtime when compared to LCEarly and LCLate is only marginally higher (less than 15 ms). Furthermore, the total runtime of LBS is merely 27 ms for a jobset size of 160. Considering the substantial performance improvement, this slight increase in runtime is justifiable. Moreover, even with a large-scale MEC and jobset size 200, the runtime of LBS remains under 100 ms, underscoring its practical viability.

VII. Conclusion

This paper addressed a deadline-constrained job mapping and resource management problem, EMJS, in MEC with both communication and computation contentions, which jointly considered job scheduling, computation resource allocation, and ED mobility. For the offline EMJS, we introduce a pseudo-polynomial approximation algorithm named LHJS with a constant approximation ratio. For the online EMJS, we propose an online scheduling algorithm named LBS with practical runtimes. Experimental results show that both LHJS and LBS significantly outperform their baseline algorithms, and highlight that allocating full server resources to each job during processing may not be an effective strategy, particularly when the server resource demand of the jobset is high.

In this work, we considered a sequential data communication model (OFDM) in the wireless network. With the recent OFDMA data communication model in 5G, the subcarriers in each wireless channel can be assigned to different users simultaneously, leading to a similar resource allocation model as the server. In future work, we plan to explore approximation algorithms for job scheduling in the OFDMA-enabled MEC.

REFERENCES

[1] W. Ji, T. Ebrahimi, Z. Li, J. Yuan, D. O. Wu, and Y. Xin, "Guest editorial: Emerging visual iot technologies for future communications and networks," *IEEE Wireless Communications*, vol. 28, no. 4, pp. 10–11, 2021.

[2] M. Aazam, S. Zeadally, and K. A. Harras, "Fog computing architecture, evaluation, and future research directions," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 46–52, 2018.

[3] M. Schneider, J. Rambach, and D. Stricker, "Augmented reality based on edge computing using the example of remote live support," in *2017 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2017, pp. 1277–1282.

[4] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *2012 IEEE symposium on computers and communications (ISCC)*. IEEE, 2012, pp. 000 059–000 066.

[5] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58 443–58 469, 2020.

[6] C. Ma and Y. Yang, "A battery-aware scheme for routing in wireless ad hoc networks," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 8, pp. 3919–3932, 2011.

[7] K. Shuai, Y. Miao, K. Hwang, and Z. Li, "Transfer reinforcement learning for adaptive task offloading over distributed edge clouds," *IEEE Transactions on Cloud Computing*, 2022.

[8] B. Ma, Z. Ren, W. Cheng, J. Wang, and W. Zhang, "Latency-constrained multi-user efficient task scheduling in large-scale internet of vehicles," *IEEE Transactions on Mobile Computing*, 2024.

[9] X. Zhang, T. Lin, C.-K. Lin, Z. Chen, and H. Cheng, "Computational task offloading algorithm based on deep reinforcement learning and multi-task dependency," *Theoretical Computer Science*, p. 114462, 2024.

[10] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 668–682, 2019.

[11] I. Sorkhoh, D. Ebrahimi, R. Atallah, and C. Assi, "Workload scheduling in vehicular networks with edge cloud capabilities," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 9, pp. 8472–8486, 2019.

[12] Y. Xu, L. Chen, Z. Lu, X. Du, J. Wu, and P. C. K. Hung, "An adaptive mechanism for dynamically collaborative computing power and task scheduling in edge environment," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 3118–3129, 2023.

[13] J. Lou, Z. Tang, S. Zhang, W. Jia, W. Zhao, and J. Li, "Cost-effective scheduling for dependent tasks with tight deadline constraints in mobile edge computing," *IEEE Transactions on Mobile Computing*, 2022.

[14] Y. Gao, J. Tao, H. Wang, Z. Wang, W. Sun, and C. Song, "Joint server deployment and task scheduling for the maximal profit in mobile edge computing," *IEEE Internet of Things Journal*, 2023.

[15] S. Abdi, M. Ashjaei, and S. Mubeen, "Task offloading in edge-cloud computing using a q-learning algorithm," in *The International Conference on Cloud Computing and Services Science, CLOSER (2024)*, 2024.

[16] Y. Sang, J. Wei, Z. Zhang, and B. Wang, "A mobility-aware task scheduling by hybrid pso and ga for mobile edge computing," *Cluster Computing*, pp. 1–16, 2024.

[17] M. Tiwari, I. Maity, and S. Misra, "Rate: Reliability-aware task service in fog-enabled iov environments," *IEEE Transactions on Cognitive Communications and Networking*, 2024.

[18] T. Zhu, T. Shi, J. Li, Z. Cai, and X. Zhou, "Task scheduling in deadline-aware mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4854–4866, 2019.

[19] X. Huang and G. Huang, "Joint optimization of energy and task scheduling in wireless-powered irs-assisted mobile-edge computing systems," *IEEE Internet of Things Journal*, vol. 10, no. 12, pp. 10 997–11 013, 2023.

[20] L. Huang and Q. Yu, "Mobility-aware and energy-efficient offloading for mobile edge computing in cellular networks," *Ad Hoc Networks*, vol. 158, p. 103472, 2024.

[21] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2287–2295.

[22] D. Han, W. Chen, and Y. Fang, "Joint channel and queue aware scheduling for latency sensitive mobile edge computing with power constraints," *IEEE Transactions on Wireless Communications*, vol. 19, no. 6, pp. 3938–3951, 2020.

[23] L. Pu, X. Chen, G. Mao, Q. Xie, and J. Xu, "Chimera: An energy-efficient and deadline-aware hybrid edge computing framework for vehicular crowdsensing applications," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 84–99, 2019.

[24] S. Pradhan, S. Tripathy, and R. Matam, "Towards optimal edge resource utilization: Predictive analytics and reinforcement learning for task offloading," *Internet of Things*, p. 101147, 2024.

[25] N. Chauhan and R. Agrawal, "A probabilistic deadline-aware application offloading in a multi-queueing fog system: A max entropy framework," *Journal of Grid Computing*, vol. 22, no. 1, pp. 1–22, 2024.

[26] W. Li, S. Li, H. Shi, W. Yan, and Y. Zhou, "Uav-enabled fair offloading for mec networks: a drl approach based on actor-critic parallel architecture," *Applied Intelligence*, pp. 1–18, 2024.

[27] X. Lai, T. Wu, C. Pan, L. Mai, and A. Nallanathan, "Short-packet edge computing networks with execution uncertainty," *IEEE Transactions on Green Communications and Networking*, 2024.

[28] L. Ben Yamin, J. Li, K. Sarpatwar, B. Schieber, and H. Shachnai, "Maximizing throughput in flow shop real-time scheduling," *Leibniz international proceedings in informatics*, vol. 176, no. 48, 2020.

[29] B. Jamil, H. Ijaz, M. Shojafar, K. Munir, and R. Buyya, "Resource allocation and task scheduling in fog computing and internet of everything environments: A taxonomy, review, and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–38, 2022.

[30] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2131–2165, 2021.

[31] S. Ramanathan, N. Shivaraman, S. Suryasekaran, A. Easwaran, E. Borde, and S. Steinhorst, "A survey on time-sensitive resource allocation in the cloud continuum," *it-Information Technology*, vol. 62, no. 5-6, pp. 241–255, 2020.

[32] Intel, "Different wi-fi protocols and data rates," 2023. [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000005725/wireless/legacy-intel-wireless-products.html

[33] M. M. Hamdi and M. S. Abood, "Transmission over ofdm and sc-fdma for lte systems," in *Intelligent Systems Design and Applications*, A. Abraham, V. Piuri, N. Gandhi, P. Siarry, A. Kaklauskas, and A. Madureira, Eds. Cham: Springer International Publishing, 2021, pp. 722–731.

[34] Z. E. Ankaralı, B. Peköz, and H. Arslan, "Enhanced ofdm for 5g ran," *ZTE Communications*, vol. 15, no. S1, pp. 11–20, 2020.

[35] Y. G. Li and G. L. Stuber, *Orthogonal frequency division multiplexing for wireless communications*. Springer Science & Business Media, 2006.

[36] R. Zhou, J. Pang, Q. Zhang, C. Wu, L. Jiao, Y. Zhong, and Z. Li, "Online scheduling algorithm for heterogeneous distributed machine learning jobs," *IEEE Transactions on Cloud Computing*, 2022.

[37] X. Fang, J. Huang, F. Wang, L. Zeng, H. Liang, and H. Wang, "Constgat: Contextual spatial-temporal graph attention network for travel time estimation at baidu maps," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 2697–2705. [Online]. Available: https://doi.org/10.1145/3394486.3403320

[38] T. J. Xia and G. A. Wellbrock, "Commercial 100-gbit/s coherent transmission systems," *Optical Fiber Telecommunications*, pp. 45–82, 2013.

[39] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[40] Q. Wang, M. Hempstead, and W. Yang, "A realistic power consumption model for wireless sensor network devices," in *2006 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, vol. 1, 2006, pp. 286–295.

[41] P. M. Vaidya, "An algorithm for linear programming which requires o(((m+n)n2+(m+n)1.5n)l) arithmetic operations," in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '87. New York, NY, USA: Association for Computing Machinery, 1987, p. 29–38. [Online]. Available: https://doi.org/10.1145/28395.28399

[42] R. Bar-Yehuda, M. M. Halldórsson, J. S. Naor, H. Shachnai, and I. Shapira, "Scheduling split intervals," *SIAM Journal on*

Computing, vol. 36, no. 1, pp. 1–15, 2006. [Online]. Available: https://doi.org/10.1137/S0097539703437843

[43] O. contributors, "Planet dump retrieved from https://planet.osm.org," 2017. [Online]. Available: https://www.openstreetmap.org

[44] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *2018 21st international conference on intelligent transportation systems (ITSC)*. IEEE, 2018, pp. 2575–2582.

[45] "Cellmapper." [Online]. Available: https://www.cellmapper.net/map

[46] Nvidia, "Nvidia jetson nano," 2024. [Online]. Available: https://www.nvidia.com/en-sg/autonomous-machines/embedded-systems/jetson-nano/product-development/