Online Approximation Scheme for Scheduling Heterogeneous Utility Jobs in Edge Computing

Chi Zhang, Haisheng Tan, *Senior Member, IEEE*, Haoqiang Huang, Zhenhua Han, Shaofeng H.-C. Jiang, Guopeng Li, and Xiang-Yang Li, *Fellow, IEEE*, *ACM*

***Abstract*— Edge computing systems typically handle a wide variety of applications that exhibit diverse degrees of sensitivity to job latency. Therefore, a multitude of utility functions of the job response time need to be considered by the underlying job dispatching and scheduling mechanism. Nonetheless, previous studies in edge computing mainly focused on optimizing a single utility function across all jobs, *e.g.*, linear, sigmoid, or the hard deadline. In this paper, we design online job dispatching and scheduling strategies in which different jobs can be categorized by different non-increasing utility functions. Our goal is to maximize the total utility of all scheduled jobs. We first prove that no online deterministic algorithm could achieve a competitive ratio better than the lower bound $\Omega(\frac{1}{\sqrt{\epsilon}})$ under the $(1+\epsilon)$-speed augmentation model. We proceed to propose an online algorithm, named as `O4A`, for handling jobs with heterogeneous utilities. We prove that `O4A` is $O(\frac{1}{\epsilon^2})$-competitive. We also design its distributed version, *i.e.*, `DO4A`. We implement `O4A` and `DO4A` on an edge computing testbed running deep learning inference jobs. With the production trace from Google Cluster, our experimental and large-scale simulation results indicate that `O4A` can increase the total utility by up to 50% compared with state-of-the-art methods. Besides, the performance loss of `DO4A` is only 2% compared with `O4A` with a small communication overhead involved. Moreover, both of our algorithms are robust to estimation errors in job processing time and transmission delay.**

***Index Terms*— Approximation algorithms, scheduling, edge computing.**

Manuscript received 24 November 2021; revised 22 May 2022 and 30 June 2022; accepted 11 July 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor B. Ji. Date of publication 5 August 2022; date of current version 16 February 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB0803400, in part by the NSFC under Grant 62132009, in part by the Key Research Program of Frontier Sciences (CAS) under Grant QYZDYSSW-JSC002, in part by the Fundamental Research Funds for the Central Universities at China, and in part by Peking University. A preliminary version of this work titled "Online Dispatching and Scheduling of Jobs with Heterogeneous Utilities in Edge Computing" was published in Proc. of the 21st International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (Mobihoc 2020), Boston, MA, USA, October, 2020 [DOI: 10.1145/3397166.3409122]. *(Corresponding authors: Haisheng Tan; Xiang-Yang Li.)*

Chi Zhang, Haisheng Tan, Guopeng Li, and Xiang-Yang Li are with the LINKE Laboratory and the CAS Key Laboratory of Wireless-Optical Communications, University of Science and Technology of China (USTC), Hefei 230027, China (e-mail: gzhnciha@mail.ustc.edu.cn; hstan@ustc.edu.cn; guopengli@mail.ustc.edu.cn; xiangyangli@ustc.edu.cn).

Haoqiang Huang is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China (e-mail: haoqiang.huang@connect.ust.hk).

Zhenhua Han is with Microsoft Research Asia (MSRA), Shanghai 200232, China (e-mail: hzhua201@gmail.com).

Shaofeng H.-C. Jiang is with the Center on Frontiers of Computing Studies, Peking University, Beijing 100871, China (e-mail: shaofeng.jiang@pku.edu.cn).

Digital Object Identifier 10.1109/TNET.2022.3193381

I. INTRODUCTION

EMERGING applications in the era of 5G, such as virtual/augmented reality and autonomous driving, require low-latency access to powerful computation resources [2], [3]. Edge computing is a promising technology by deploying servers at the Internet edge. In essence, the computing paradigm where mobile applications offload their latency-sensitive jobs to nearby edge servers can greatly extend the capability of mobile devices and enlarge the coverage of cloud computing.

In order to efficiently utilize the limited resources in edge-clouds, one fundamental challenge lies in job dispatching and scheduling, *i.e.*, to decide 1) onto which edge server or the remote cloud each job should be dispatched, and 2) in which order jobs should be executed on a server. Dispatching and scheduling are typically multi-objective, *e.g.*, maximizing the utilization of edge servers, maximizing the revenue of the service providers, and/or minimizing the peak resource demand. Most notably, the job response time (JRT, defined as the interval between the job release and the arrival of the computation result at the mobile device) is a principal criterion for evaluating the QoS of latency-sensitive jobs in edge computing. Therefore, we here focus on the problem of maximizing the aggregated utility of all jobs with respect to the JRT.

In edge computing, mobile applications from various users might exhibit different levels of latency sensitivity. Accordingly, resource allocation amounts to dispatching and scheduling computational jobs with heterogeneous utility functions of the JRT. Fig. 1 illustrates such an edge computing system. Multiple small-scale edge servers are geographically dispersed and inter-connected via high-speed local or metropolitan area networks. Mobile devices can reach nearby edge servers with low network latency, but can also deploy jobs to the remote cloud via the Internet while suffering from larger transmission delays. Different kinds of jobs arrive at the edge servers, which employ diverse utility functions such as: linear, hard-deadline (All-or-Nothing), and functions in-between, *e.g.*, sigmoid. Previous works [4]–[7] were mainly dedicated to optimizing a single utility function among all jobs. Nevertheless, varying the utility function will differentiate scheduling disciplines, and utility-agnostic schedulers might fail to maximize the aggregate utility of all jobs. In addition, as edge servers are typically scattered in different geographic locations, an efficient distributed scheduler is admired.

In this paper, we study online dispatching and scheduling of jobs with heterogeneous utility functions in edge-cloud

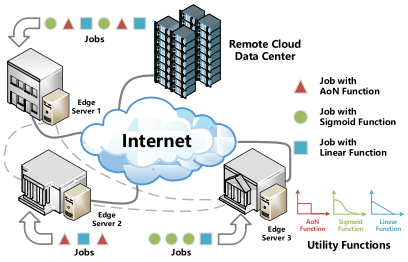


Fig. 1. Illustration of an edge-cloud system with mobile devices, edge servers, the remote cloud, alongside jobs with heterogeneous utility functions.

systems. We consider a general setting, where a set of online jobs may arrive at arbitrary time in arbitrary order. Each job is associated with a utility function, and different jobs are allowed to have various utility functions. We adopt the unrelated machine model, *i.e.*, each job has machine-dependent processing time on each server, and there is no relationship between the job processing time on different servers. Our goal is to maximize the total utility for all jobs. We use the *competitive ratio*, the ratio of the offline optimal to our online algorithm performance, as a metric to theoretically evaluate our method. In [8], it was proven that no online algorithm for the job dispatching and scheduling problem under the unrelated machine model could have a bounded competitive ratio, even when the utility functions are all linear. Therefore, we adopt the *speed augmentation model* [9] by allowing the edge servers to be $(1+\epsilon)$ times as fast as in the optimal offline algorithm, where ϵ is a small positive constant .[1] Our main contributions are summarized as:

- We formulate a general online job dispatching and scheduling problem in edge computing, where co-existing jobs may employ heterogeneous utility functions. Additionally, the upload/download delay and the unrelated-machine job processing time are considered (Sec. III).
- For the problem hardness, we give a lower bound of the competitive ratio of all online deterministic algorithms as $\Omega(\frac{1}{\sqrt{\epsilon}})$ under the $(1+\epsilon)$-speed augmentation (Theorem 1).
- We further propose an online algorithm, named O4A (standing for "One algorithm for All utility functions"), and prove its competitive ratio as $O(\frac{1}{\epsilon^2})$. We also devise its distributed version, *i.e.*, DO4A, which can achieve similar performance with an efficient querying scheme to decrease the message complexity dramatically (Sec. IV).
- We implement both O4A and DO4A on a small-scale testbed consisting of 20 edge devices. Our experiments show that O4A can increase the aggregate job utility by up to 50% compared with state-of-the-art baselines. Besides, the gap between O4A and the optimal solution is within 12%. Furthermore, the performance loss of DO4A is only 2% compared with O4A, even when one mobile device may communicate with a maximum number of 5 servers (Sec. V).

[1]In practice, the speed augmentation analysis can be understood as follows: the algorithm will achieve the theoretical performance as long as the capability of servers is upgraded by a small factor ϵ.

- Based on extensive large-scale simulations on the production trace from Google, we demonstrate that both O4A and DO4A consistently outperform the baselines over various workloads and parameters. Moreover, O4A and DO4A are robust to estimation errors in job processing time and communication delay (Sec. V).

II. Motivation

A. Heterogeneous Utilities Co-Existing in Edge Applications

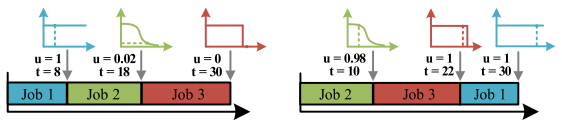
Latency is a critical criterion in edge applications, where various job requests typically exhibit different sensitivity to job response latency. We here adopt *autonomous driving* to demonstrate the coexistence of heterogeneous utilities. Autonomous driving can make use of edge computing to support demands with heterogeneous latency sensitivity, such as:

- **Obstacle Detection**: Autonomous vehicles need to collect and analyze data from multiple sensors to detect possible obstacles around vehicles. These tasks are deadline-critical that impose strict requirements on the response latency, *e.g.*, 100 ms as mentioned in [10]. Its utility function can be represented with an All-or-Nothing (AoN) function (as shown in Fig. 1).
- **Driver Assistance**: To improve driving safety, machine learning techniques are used for assisting drivers to avoid potential risks, which include fine-grained face recognition [11], body pose estimation [12], semantic scene perception, and driving state prediction [13]. The utility function could be sigmoid-like, whose QoE is sensitive to a range of latency [14].
- **Data Preprocessing**: Autonomous vehicles are estimated to generate as much as 5 terabytes of data per hour [15]. Data preprocessing on the edge (*e.g.*, compressing and filtering) can be adopted to reduce the amount of data transferred to the cloud data centers. Job completion time is commonly used to measure its efficiency, which can be modeled as optimizing a linear utility function.

In addition to autonomous driving, public edge-clouds might need to support a wide range of applications, *e.g.*, content delivery, video streaming, and IoT analytics, which can employ various sensitivity to latency. To efficiently utilize limited edge resources, edge schedulers should allow edge applications to express their utility functions w.r.t. response latency, and exploit the heterogeneity when scheduling jobs.

B. Inefficiency of Existing Schedulers

Existing job schedulers that can be deployed on edge servers are typically not aware of the coexistence of heterogeneous utilities of applications, *e.g.*, OnDisc [4] and Dedas [7]. We take the shortest job first (SJF) policy adopted by OnDisc [4] as the example policy. In the following example, we show that although SJF is the optimal policy on average job response, however, it is inefficient when optimizing total utility. As elaborated in Fig. 2, there are three jobs j_1, j_2 and j_3 to be scheduled, all of which arrive at time 0. Their processing time is $p_1 = 8, p_2 = 10$ and $p_3 = 12$, respectively. Their utility functions are $g_1(t) = 1, g_2(t) = \frac{1}{1+e^{t-14}}$ and $g_3(t) =$



(a) SJF Scheduler (b) Optimal

Fig. 2. Inefficiency of the SJF when heterogeneous utility functions coexist.

$\begin{cases} 1 & t \leq 25 \\ 0 & t > 25 \end{cases}$, which are constant function, sigmoid function and AoN function, respectively. SJF will schedule them in the order of j_1, j_2, j_3 as shown in Fig. 2 (a) with a total utility of 1.02. The optimal solution (shown in Fig. 2 (b)) is in the order of j_2, j_3, j_1 with a total utility of 2.98. Similarly, the strategy optimized for deadline-sensitive jobs, *e.g.*, earliest deadline first (EDF), may prioritize deadline-sensitive jobs too much and hurt the utilities of other jobs. In summary, heterogeneous utility functions can appear simultaneously in edge applications, while existing methods could not handle them efficiently. In this work, we try to shed a light on this problem with algorithm design and system implementation.

III. System Model

A. Edge System

We consider an edge-cloud system with a set of servers denoted as $\mathcal{S}$, including a number of heterogeneous edge servers and a remote cloud regarded as a special server with long transmission latency but abundant computation capability. We denote $\mathcal{J}$ as the set of jobs released from mobile devices. In mobile application scenarios, mobile devices and jobs can appear in arbitrary order and time, and one may assume no prior information before the job release. To avoid migration overhead, a server cannot migrate jobs to the others after the job dispatching. Preemption is allowed so that an executing job might be halted and resumed later.

1) Jobs: When a job $i \in \mathcal{J}$ is dispatched to server $j \in \mathcal{S}$ after its release time denoted as r_i, there is an upload delay $\Delta^{\uparrow}_{i,j}$ to transmit its initial data. Similarly, there is a download delay $\Delta^{\downarrow}_{i,j}$ to transmit the processing result from the server to the mobile device. Therefore, job i cannot be ready to process until time $r_i + \Delta^{\uparrow}_{i,j}$, *i.e.*, the arrival time of job i on server j. In addition, the job completion time is when job i completes its processing at server j. Here, we adopt the unrelated machine model. That is to say, the processing time of job i on server j, denoted as $p_{i,j}$, is not identical or related to its processing time on other servers. We also assume jobs are independent of each other. Finally, the job finish time, denoted as f_j, is when the computation result arrives at the mobile device after a download delay. The job response time (JRT) of i, denoted as c_i, is defined as the interval between its release and finish, *e.g.*, $c_i = f_i - r_i$. Since the cloud can be modeled as a special server, we do not differentiate the cloud and the edge server explicitly here. Note that the data transmission need not take over the computation resource of servers. Thus, the server can process other jobs during data transmission.

2) Utility Function: Each job i is released and associated with a utility function with respect to its JRT, denoted as $g_i(c_i)$. As stated above, we consider heterogeneous utility functions for our jobs. Different jobs could have different utility functions to indicate their sensitivity to the JRT. Generally, a utility function can be any non-increasing function, which means we cannot gain a higher utility by postponing the job's processing. We also assume $g_i(c_i) \geq 0, \forall c_i$, *i.e.*, completing a job will never bring negative utility to the system.

B. Problem Formulation

With the aforementioned model, our problem is to design online dispatching and scheduling strategies for jobs with heterogeneous utilities to maximize the total utility, *i.e.*, $\sum_{i \in \mathcal{J}} g_i(c_i)$.

Problem 1:

$$\max_{\{x_{i,j}(t)\},\{y_{i,j}\},\{f_i\}} \sum_{i \in \mathcal{J}} g_i(c_i), \tag{1}$$

$$\text{s.t.} \quad x_{i,j}(t) \in \{0,1\}, \forall i \in \mathcal{J}, j \in \mathcal{S}, t \geq r_i + \Delta^{\uparrow}_{i,j} \tag{2}$$

$$y_{i,j} \in \{0,1\}, \quad \forall i \in \mathcal{J}, j \in \mathcal{S} \tag{3}$$

$$\sum_{i \in \mathcal{J}} x_{i,j}(t) \leq 1, \quad \forall j \in \mathcal{S}, t \geq r_i + \Delta^{\uparrow}_{i,j} \tag{4}$$

$$\sum_{j \in \mathcal{S}} y_{i,j} = 1, \forall i \in \mathcal{J} \tag{5}$$

$$f_i \in \left\{ z \middle| \int_{r_i + \Delta^{\uparrow}_{i,j}}^{z - \Delta^{\downarrow}_{i,j}} x_{i,j}(t) y_{i,j} dt \geq p_{i,j}, \forall j \in \mathcal{S} \right\}, \quad \forall i \in \mathcal{J}, \tag{6}$$

where $y_{i,j}$ is a binary variable that equals to 1 if the job i is dispatched to the server j, and 0 otherwise. $x_{i,j}(t)$ is a binary variable that equals to 1 if the server j is processing job i at time t, and 0 otherwise. Eqn. (1) is the objective function that maximizes the total utility of all jobs. Constraint (4) and (5) guarantee each server only processes one job at any time, and each job is dispatched to exactly one server, respectively. Constraint (6) guarantees each job is processed for at least $p_{i,j}$ time if it is finished.

Specifically, we study Problem 1 in an online mode. $x_{i,j}(t)$ and $y_{i,j}$ can only be determined after time r_i. The value of $p_{i,j}$, $\Delta^{\uparrow}_{i,j}$, $\Delta^{\downarrow}_{i,j}$ and function $g_i(t)$ are agnostic before time r_i. We use the competitive ratio to analyze the online algorithm:

Definition 1 (Competitive Ratio): An online algorithm is c-competitive if for any job set, we have $\frac{OPT}{ALG} \leq c$, where ALG *denotes the utility gained by the online algorithm and* OPT *denotes the total utility of the offline optimal solution.*

C. Hardness

Problem 1 has been proved to be hard to achieve a bounded competitive ratio even in its simplified version with hard-deadline utility function across all jobs [16], we employ the speed augmentation model in our analysis. Formally, we define $(1+\epsilon)$-speed augmentation [9]:

Definition 2 (Speed Augmentation): A sever with $(1+\epsilon)$-speed augmentation means that any job i only takes $\frac{p_i}{1+\epsilon}$ processing time, where p_i is job's processing time on the original sever.

For a constant c, $(1+\epsilon)$-speed c-competitive is defined as:

Definition 3 (Competitive Ratio with Speed Augmentation): An online algorithm is $(1+\epsilon)$-speed c-competitive if for any job set, we have $\frac{OPT}{ALG} \leq c$, where ALG *denotes the utility gained by the online algorithm with $(1+\epsilon)$-speed and* OPT *denotes the total utility of the offline optimal solution on the original servers.*

In this part, we prove the hardness of Problem 1 in the speed augmentation model.

Theorem 1: For every $\epsilon \in (0, \frac{1}{2})$, all deterministic online algorithms for Problem 1 are $(1+\epsilon)$-speed $\Omega(\frac{1}{\sqrt{\epsilon}})$-competitive.

Proof: The hard instance is constructed as follows. We can only take a single server in consideration and neglect the upload and download delay. Then we study two types of jobs:

- Job j_1: a big job with p processing time. Its utility function is $g_1(t) = \begin{cases} p, t \leq p \\ 0, t > p \end{cases}$.
- Job j_2: a small job with δ processing time. Its utility function is $g_2(t) = \begin{cases} \tau\delta, t \leq \delta \\ 0, t > \delta \end{cases}$, where $\tau = \sqrt{\frac{1+\epsilon}{\epsilon(1-\epsilon)}}$.

Note that $\tau\delta$ is negligible compared to p. We release one j_1 at time 0 and release j_2 one by one after time 0. Before time $\frac{1-\epsilon}{1+\epsilon}p$, once the algorithm spends more than $\frac{\epsilon}{1+\epsilon}p$ time in processing j_2, we stop releasing j_2. Then at time $\frac{1-\epsilon}{1+\epsilon}p - \delta$, we face two cases.

Case 1: the algorithm spends more than $\frac{\epsilon}{1+\epsilon}p$ time on j_2. Since j_1 cannot complete before time p even with $(1+\epsilon)$ speed augmentation, the utility gained by the algorithm is $\epsilon\tau p$. Because the adversary can process j_1 only, it will earn the utility p, thus the ratio between the adversary's utility and the algorithm's utility is $\frac{p}{\epsilon\tau p} = \sqrt{\frac{1-\epsilon}{1+\epsilon} \times \frac{1}{\epsilon}}$.

Case 2: the algorithm spends no more than $\frac{\epsilon}{1+\epsilon}p$ time on j_2. We first keep releasing j_2. At time $\frac{1-\epsilon}{1+\epsilon}p - \delta$, we stop releasing j_2 and release one j_1. Since we can only guarantee that one j_1 can complete with p utility, the utility gained by the algorithm is at most $(1+\epsilon\tau)p$. However, the adversary can only process j_2 before time $\frac{1-\epsilon}{1+\epsilon}p - \delta$ and start processing the last j_1 at its arrival, the utility gained by the adversary is $(\frac{1-\epsilon}{1+\epsilon}\tau + 1)p$. We have the ratio between the adversary's utility and the algorithm's utility is $\frac{(\frac{1-\epsilon}{1+\epsilon}\tau+1)p}{(1+\epsilon\tau)p} = \sqrt{\frac{1-\epsilon}{1+\epsilon} \times \frac{1}{\epsilon}}$. □

IV. The O4A Algorithm

In this section, we present our online algorithm O4A, to solve the joint dispatching and scheduling problem in edge systems featuring jobs with heterogeneous utility functions.

A. General Idea

For each job dispatched to a server, O4A calculates its *tentative* execution time and its *tentative* completion time, so as to give a guide to conservatively make a scheduling plan. In order to address the impact of potential preemption by future jobs, when scheduling each job on a server, O4A adopts a parameter to reserve more execution time. Given a job's utility function, O4A calculates the potential time intervals on its dispatched server that can be used to execute this job. At any time, O4A picks a job following the highest utility density principle, defined as the potential utility divided by the execution time. Moreover, when dispatching a job after its release, O4A will select the edge server that can maximize the job's utility myopically, *i.e.*, by assuming there will be no future jobs. We next elaborate O4A in detail.

B. Tentative Execution

O4A leverages the speed augmentation parameter ϵ when making scheduling decisions. Assuming the job i is dispatched to server j, we define the job's *tentative execution time* as $p^\dagger_{i,j} = \frac{1+\beta\epsilon}{1+\epsilon}p_{i,j}$, which is a conservative processing time that O4A allocates to the job i. Note that an online server with speed augmentation $(1+\epsilon)$, can complete job i after $\frac{p_{i,j}}{1+\epsilon}$ time units; the extra $\frac{\beta\epsilon}{1+\epsilon}p_{i,j}$ in $p^\dagger_{i,j}$ is the time reserved by O4A for addressing potential preemption by future-arriving jobs. When planning the job's execution, O4A tentatively takes $p^\dagger_{i,j}$ as the job's processing time, and thus determines its *tentative completion time* (denoted by $d^\dagger_{i,j}$), *i.e.*, the time when the job i is processed for a duration of $p^\dagger_{i,j}$.

C. Computing a Tentative Execution Plan

Here, we elaborate in Algorithm 1 how we calculate job i's tentative execution plan given that it is dispatched to server j. We define $\mathcal{I}_{i,j}$ as job i's executable time, *i.e.*, the time when server j schedules job i. If $t \notin \mathcal{I}_{i,j}$, server j will not schedule job i at time t. Normally, the number of distinct utility density is approximately equal to the number of queued jobs. To reduce the positions where new jobs may be inserted, we use *logarithmic discretization* to round the original utility functions $g_i(t)$ to $g^D_i(t)$ (Line 2–2). As we will prove later, this optimization does not hurt the asymptotic performance of O4A, while the possible positions will be reduced to the logarithm of the ratio of the maximum density to the minimum density. We define the utility density of a job as its potential utility under its tentative scheduling plan divided by its real processing time (Line 6). In each server, O4A schedules jobs following the highest utility density-first principle over all unfinished jobs. Since a job's utility density depends on its tentative completion time $d^\dagger_{i,j}$, Algorithm 1 computes the minimum time $d^\dagger_{i,j}$ for job i, together with its executable time $\mathcal{I}_{i,j}$ and its utility density $u_{i,j}$ (Line 8–21).

We enumerate the distinct utility density on server j in ascending order, so as to compute the priority assigned to job i. For the γ-th utility density u_γ, we calculate job i's tentative completion time if job i is assigned with the γ-th priority (Line 13–16). The case $\gamma = 0$ corresponds to the case that i has the lowest priority, and thus it cannot preempt any other jobs (Line 10–11). In the case $\gamma > 0$, we want to find out all time intervals not occupied by jobs with a utility density higher than u_γ, *i.e.*, $\mathcal{J}^\gamma_Q$. To achieve this, we use *segment tree optimization*, maintaining a segment tree for each γ to store all intervals with a utility density greater than u_γ. After establishing the segment trees, we can use binary search to determine $d^\gamma_{i,j}$. Since in the online scenario, the timeline will be infinitely extended, we adopt *timeline scrolling* periodically, which avoids frequent deletions on the segment trees caused by job completion.

We define α as a preemption threshold: job i is placed at the γ-th position when its utility density is higher than α times

Algorithm 1 Tentative Schedule Planning

1 **Input** *job i, server j*;
2 $\alpha = 2 + \frac{2}{\epsilon}$;
3 $\lambda_i^{\dagger}(t) = \arg\max_{\lambda}\{\lambda | p_{i,j} \cdot 2^{\lambda} \leq g_i(t)\}$;
4 $g_i^D(t) = p_{i,j} \cdot 2^{\lambda_i^{\dagger}(t)}$;
5 $\mathcal{J}_Q$ = the set of unfinished jobs at server j;
6 Define $u_{i',j} = g_{i'}^D(d_{i',j}^{\dagger} + \Delta_{i',j}^{\downarrow} - r_i)/p_{i',j}$ as the utility density of job $i' \in \mathcal{J}_Q$;
7 Let the distinct utility density in $\mathcal{J}_Q$ be $u_1, u_2, \ldots, u_K$ (in ascending order);
8 The set of feasible tentative density $\mathcal{U} = \varnothing$;
9 **for** $\gamma = 0$ *to* K **do**
10 **if** $\gamma = 0$ **then**
11 $\mathcal{I}_{i,j}^{\gamma} = \{t : t \geq r_i + \Delta_{i,j}^{\uparrow}, t \notin \cup_{i' \in \mathcal{J}_Q} \mathcal{I}_{i'}\}$;
12 **else**
13 i^{γ} = the γ-th job in $\mathcal{J}_Q$;
14 $\mathcal{J}_Q^{\gamma} = \{i' : i' \in \mathcal{J}_Q, u_{i',j} > u_{\gamma}\}$;
15 $\mathcal{I}_{i,j}^{\gamma} = \{t : t \geq r_i + \Delta_{i,j}^{\uparrow}, t \notin \cup_{i' \in \mathcal{J}_Q^{\gamma}} \mathcal{I}_{i'}\}$;
16 $d_{i,j}^{\gamma} = \min\{t | \int_{r_i}^{t} \mathbf{1}[t \in \mathcal{I}_{i,j}^{\gamma}]dt \geq p_{i,j}^{\dagger}\}$;
17 $u_{i,j}^{\gamma} = g_i^D(d_{i,j}^{\gamma} + \Delta_{i,j}^{\downarrow} - r_i)/p_{i,j}$;
18 **if** $u_{i,j}^{\gamma} > \alpha u_{i',j}$ *for all* $i' \in \mathcal{J}_Q - \mathcal{J}_Q^{\gamma}$ **then**
19 Add $u_{i,j}^{\gamma}$ to $\mathcal{U}$;
20 $\gamma^{\dagger} = \arg\max_{\gamma} \mathcal{U}$;
21 $\mathcal{I}_{i,j} = \mathcal{I}_{i,j}^{\gamma^{\dagger}}$; $d_{i,j}^{\dagger} = d_{i,j}^{\gamma^{\dagger}}$; $u_{i,j} = u_{i,j}^{\gamma^{\dagger}}$;

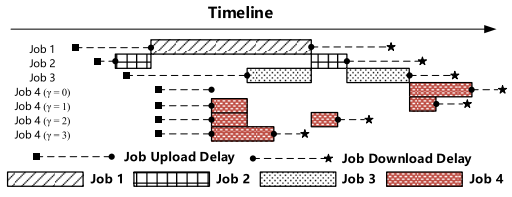


Fig. 3. Tentative schedule computation for a newly released job (Job 4).

that of any other job with lower priorities. Namely, job i can preempt jobs with lower priorities only if it has a high enough utility density (Line 17–19). We assign job i with the priority corresponding to the highest utility density (Line 20–21).

Fig. 3 illustrates how Algorithm 1 works. In the beginning, there are 3 unfinished jobs on the server: Job 1, 2 and 3 in ascending order of their utility density. Job 2 has a higher density than Job 1, but less than α times, so Job 2's executable time cannot overlap with Job 1's executable time. Similarly, Job 3's executable time cannot overlap with Job 2's. As Job 3's utility density is higher than α times of Job 1's, Job 3' executable time can overlap with Job 1's so that Job 3 can preempt Job 1 when it arrives. When the newly released Job 4 arrives, Algorithm 1 needs to compute the tentative completion time by enumerating its potential priorities. When $\gamma = 0$, Job 4 has the lowest priority so that it can only start processing when all the other three jobs finish their computation. When $\gamma = 1$, Job 4 may preempt Job 1 by overlapping its executable time with Job 1's, but still needs to avoid overlapping with Job 2 and 3. When $\gamma = 2$, and $\gamma = 3$, Job 4 can further consider preempting Job 2 and Job 3, respectively. Job 4 will have different tentative completion time, utility density, and executable time when it is placed on different priorities. O4A will choose the one that maximizes its utility.

Algorithm 2 Dispatching and Scheduling Policy in O4A

1 **Job Dispatching:** When job i is released at time r_i, it is dispatched to server $j = \arg\min_{j' \in \mathcal{S}}\{d_{i,j'}^{\dagger} + \Delta_{i,j'}^{\downarrow}\}$;
2 Determine the tentative finish time of job i: $f_i = d_{i,j}^{\dagger} + \Delta_{i,j}^{\downarrow}$;
3 **Job Scheduling:** At time t, server j schedules the executable job i with the highest utility density, *i.e.*, $i = \arg\max_{\{i' | t \in \mathcal{I}_{i',j}\}} u_{i',j}$;

D. Job Scheduling & Dispatching

Based on O4A's tentative schedule planning, we elaborate in Algorithm 2 how O4A dispatches and schedules newly arriving jobs. When released, a job is dispatched to the server that yields the highest utility under the tentative scheduling (Line 1). Next, the tentative time of job i is calculated (Line 2). On each server, O4A picks the executable job with the highest utility density (a job is executable at the time t if and only if $t \in \mathcal{I}_{i,j}$) (Line 3). Note that, a job's utility density and executable time are decided by Algorithm 1 when its dispatching decision is made and does not change afterward.

Time Complexity: Let K be the number of distinct utility density after discrete optimization, which is the logarithm of the ratio of the maximum density to the minimum density. Let L be the maximum number of intervals among all jobs. Let T be the length of the timeline. The time complexity to dispatch each job in O4A is $O(K \cdot L \cdot \log^2 T)$.

E. Competitive Analysis

We employ the speed-augmentation model, which means that we compare the utility gained by our algorithm on servers with $(1+\epsilon)$ speed with the utility gained by an offline optimal solution on original servers. By convention, we let *ALG* be the total utility that O4A gets and let *OPT* denote the total utility earned by the optimal solution computed offline. Let OPT^D be the optimal utility after discretization. Our analysis is inspired by [17], and comprises three steps:

- Although O4A might not finish all jobs by the tentative finish time, we lower-bound *ALG* by the total tentative utility under the assumption that all jobs finish before their tentative finish time (Lemma 1). The high-level idea is to use the charging scheme to transfer utility from the highest density job to other jobs in the solution of O4A, so that all the jobs, including those finished later than their tentative finish time, have a comparable utility to their tentative utility.
- In Lemma 2 and Lemma 3, we upper-bound OPT^D by the cumulative tentative utility. To achieve this, we need to prove for a fixed server j, the total utility of the jobs O4A dispatched to server j is comparable to the total utility of the optimal on server j, by calculating the relationship of the total length of processing time of jobs whose density is at least u of the optimal and O4A.

- By combining Lemma 1, Lemma 2 and Lemma 3 we can bound *ALG* by *OPT* , from which we get Theorem 2.

For ease of notation, we let $f_i \triangleq d_{i,j}^{\dagger} + \Delta_{i,j}^{\downarrow}, \mathcal{I}_i \triangleq \mathcal{I}_{i,j}$ be the tentative finish time and tentative interval of job i if job i is dispatched to server j, respectively. We let $w_i^D \triangleq g_i^D(f_i - r_i)$ be the discretized tentative utility, and let $u_i^D \triangleq \frac{w_i^D}{p_{i,j}}$ be the discretized tentative density. Finally, we denote the set of all jobs by $\mathcal{J}$, and use $\mathcal{C}$ for the set of jobs that finish prior to their tentative finish time as calculated by O4A.

Lemma 1: The utility obtained from O4A is at least $(1 - \frac{1+\beta\epsilon}{\beta\epsilon(\alpha-1)})$ *times the total utility if each job i was completed at time f_i, i.e.,* $ALG \geq (1 - \frac{1+\beta\epsilon}{\beta\epsilon(\alpha-1)}) \sum_{i\in\mathcal{J}} w_i^D$.

Proof: For each server j, we will use the charging scheme to transfer utility among jobs. Let non-negative h_i^{in} be the utility transferred to job i and h_i^{out} be the utility transferred out from job i. For any time t and server j, let $X_{t,j}$ be the set of jobs whose tentative interval $\mathcal{I}_i$ contains t. The job i with the highest density $u_i^D = \frac{w_i^D}{p_{i,j}}$ in $X_{t,j}$ transfers utility to other jobs in $X_{t,j}$. Here, we define transfer speed as the amount of utility transferred in a unit time. For each job $i' \in X_{t,j} - \{i\}$, the transfer speed is set to $(\frac{1+\epsilon}{\beta\epsilon})\frac{w_{i'}^D}{p_{i',j}}$. Note that the total utility of all jobs will not change, that is $\sum_{i\in\mathcal{J}} h_i^{in} = \sum_{i\in\mathcal{J}} h_i^{out}$. Thus, we have $ALG = ALG + \sum_{i\in\mathcal{J}} h_i^{in} - \sum_{i\in\mathcal{J}} h_i^{out}$. By the definition of $\mathcal{C}$ and w_i^D, we have $ALG \geq \sum_{i\in\mathcal{C}} w_i^D$. For each job i that is not completed, there must be some more dense jobs processing at least $\frac{\beta\epsilon}{1+\epsilon}p_{i,j}$ units of time in interval $\mathcal{I}_i$. That is $h_i^{in} \geq \frac{1+\epsilon}{\beta\epsilon} \cdot \frac{w_i^D}{p_{i,j}} \cdot \frac{\beta\epsilon}{1+\epsilon}p_{i,j} = w_i^D,\ i \in \mathcal{J} - \mathcal{C}$. Thus, we have $\sum_{i\in\mathcal{J}-\mathcal{C}} h_i^{in} \geq \sum_{i\in\mathcal{J}-\mathcal{C}} w_i^D$.

By the definition of $\mathcal{I}_i$, for any two jobs i and i' assigned to server j where $\mathcal{I}_i$ and $\mathcal{I}_{i'}$ overlap, it must hold that either $u_i^D > \alpha u_{i'}^D$ or $u_{i'}^D > \alpha u_i^D$. Thus at any time t the speed of utility transferred from job i is at most $\frac{1+\epsilon}{\beta\epsilon} \cdot \frac{w_i^D}{p_{i,j}} \sum_{k=1}^{\infty} \frac{1}{\alpha^k} \leq \frac{1+\epsilon}{\beta\epsilon(\alpha-1)} \cdot \frac{w_i^D}{p_{i,j}}$. The total length of intervals in $\mathcal{I}_i$ is $\frac{1+\beta\epsilon}{1+\epsilon}p_{i,j}$, thus $h_i^{out} \leq \frac{1+\epsilon}{\beta\epsilon(\alpha-1)} \cdot \frac{w_i^D}{p_{i,j}} \cdot \frac{1+\beta\epsilon}{1+\epsilon}p_{i,j} = \frac{1+\beta\epsilon}{\beta\epsilon(\alpha-1)}w_i^D$. That is $\sum_{i\in\mathcal{J}} h_i^{out} \leq \sum_{i\in\mathcal{J}} \frac{1+\beta\epsilon}{\beta\epsilon(\alpha-1)}w_i^D$. Finally, we have $ALG \geq \sum_{i\in\mathcal{C}} w_i^D + \sum_{i\in\mathcal{J}-\mathcal{C}} w_i^D - \frac{1+\beta\epsilon}{\beta\epsilon(\alpha-1)} \sum_{i\in\mathcal{J}} w_i^D \geq \left(1 - \frac{1+\beta\epsilon}{\beta\epsilon(\alpha-1)}\right) \sum_{i\in\mathcal{J}} w_i^D$, which concludes our proof. □

Now, we are going to upper-bound OPT^D by the sum of the tentative utility of all jobs, *i.e.*, $\sum_{i\in\mathcal{J}} w_i^D$. Let A_1^* be the set of jobs such that $i \in A_1^*$ implies that the tentative finish time f_i set by the algorithm is no later than f_i^*, *i.e.*, the finish time of job i in the optimal solution; let A_2^* comprise the remaining jobs. Besides, for a fixed server j, let $A_{1,j}^*, A_{2,j}^*$ be the subsets of A_1^*, A_2^*, respectively, which only consist of jobs dispatched to server j in the optimal solution. We have that the total utility of jobs in A_1^* at the optimal solution is less than $\sum_{i\in\mathcal{J}} w_i^D$. We next concentrate on bounding the utility of jobs in A_2^* of the optimal solution. For each server j and $u \geq 0$, let $L_j^*(u)$ denote the total processing time of jobs whose density is at least u and that is in set $A_{2,j}^*$ in the optimal solution's scheduling. Let $L_j(\frac{u}{\alpha})$ be the total length of time where the algorithm processes a job with density at least $\frac{u}{\alpha}$ on server j. For a set of intervals $\mathcal{I}$, let $L(\mathcal{I})$ be the sum of the length of all intervals in $\mathcal{I}$.

Lemma 2: For every server j and all $u > 0$, $\beta \in (0,1)$, $L_j^*(u) \leq \frac{2(1+\epsilon)}{(1-\beta)\epsilon}L_j(\frac{u}{\alpha})$.

Proof: Fix a server j. For each job i, we define an interval $[r_i + \Delta_{i,j}^{\uparrow}, f_i^* - \Delta_{i,j}^{\downarrow}]$, where r_i is the release time of job i and f_i^* is the finish time in the optimal scheduling. Let $W_j^*(u)$ be the set of all the intervals of jobs with density at least u in A_2^*. For ease of analysis, let $M_j(u)$ be the minimal subset of $W_j^*(u)$ whose intervals span every time contained in an interval in $W_j^*(u)$. Then we can divide all the intervals of $M_j(u)$ into two sets, *i.e.*, $M_j^1(u)$ and $M_j^2(u)$, each of which does not have overlapping intervals. Without loss of generality, we assume that the total length of time contained in an interval of $M_j^1(u)$ is at least that of $M_j^2(u)$. Then we have the total length of time contained in an interval of $M_j^1(u)$ is at least half of that of $W_j^*(u)$, *i.e.*, $L(M_j^1(u)) \geq \frac{1}{2}L(W_j^*(u))$.

According to the definition of tentative completion time, for each job i in A_2^*, the total length of time where the algorithm processes a job with density at least $\frac{u_i^D}{\alpha}$ in the interval $[r_i + \Delta_{i,j}^{\uparrow}, f_i^* - \Delta_{i,j}^{\downarrow}]$ is at least $\frac{(1-\beta)\epsilon}{1+\epsilon}[f_i^* - \Delta_{i,j}^{\downarrow} - (r_i + \Delta_{i,j}^{\uparrow})]$. Otherwise, the tentative completion time of job i on server j calculated by O4A will be earlier than its completion time in the optimal solution on server j, which violates the assumption that job i belongs to A_2^*. Since $M_j^1(u)$ does not have overlapping intervals, we have $L_j\left(\frac{u}{\alpha}\right) \geq \frac{(1-\beta)\epsilon}{1+\epsilon}L(M_j^1(u)) \geq \frac{(1-\beta)\epsilon}{2(1+\epsilon)}L(W_j^*(u))$. Furthermore, by definition, we have $L_j^*(u) \leq L(W_j^*(u)) \leq \frac{2(1+\epsilon)}{(1-\beta)\epsilon}L_j\left(\frac{u}{\alpha}\right)$. □

Lemma 3: $\sum_{i\in A_2^*} w_i^{D*} \leq \frac{2\alpha(1+\epsilon)}{(1-\beta)\epsilon} \sum_{i\in\mathcal{J}} w_i^D$.

Proof: Fix a server j. Let $u_i^{D*} = w_i^{D*}/p_{i,j}$ and $u_i^D = w_i^D/p_{i,j}$. Let $A_{2,j}^*(u)$ denote the set of jobs in A_2^* scheduled to server j with density at least u in the optimal scheduling. Let $A_j(u)$ be the set of all jobs scheduled to server j with density at least u in O4A. For ease of analysis, we sort the jobs in $A_{2,j}^*(0)$ from 1 to n ($|A_{2,j}^*(0)| = n$) by their densities in the optimal scheduling with a non-increasing order.

In addition, we add a virtual job $n+1$ with density 0 in the end of the sequence for the ease of notation. According to Lemma 2, for each $i \in A_{2,j}^*(0)$ we have

$$\sum_{i'\in A_{2,j}^*(u_i^{D*})} p_{i',j} = L_j^*(u_i^{D*}) \leq \frac{2(1+\epsilon)}{(1-\beta)\epsilon}L_j(u_i^{D*}/\alpha) = \frac{2(1+\epsilon)}{(1-\beta)\epsilon} \sum_{i'\in A_j(u_i^{D*}/\alpha)} p_{i',j}.$$

By using this inequality, we have

$$\begin{aligned}
&\sum_{i\in A_{2,j}^*(0)} w_i^{D*} \\
&= \sum_{i\in A_{2,j}^*(0)} p_{i,j} \sum_{i\in A_{2,j}^*(0), i'\geq i} (u_{i'}^{D*} - u_{i'+1}^{D*}) \\
&= \sum_{i\in A_{2,j}^*(0)} (u_i^{D*} - u_{i+1}^{D*}) \sum_{i'\in A_{2,j}^*(0), i'\leq i} p_{i',j} \\
&= \sum_{i\in A_{2,j}^*(0)} (u_i^{D*} - u_{i+1}^{D*}) \sum_{i'\in A_{2,j}^*(u_i^{D*})} p_{i',j}
\end{aligned}$$

$$\leq \frac{2(1+\epsilon)}{(1-\beta)\epsilon} \sum_{i\in A^*_{2,j}(0)} (u_i^{D*} - u_{i+1}^{D*}) \sum_{i'\in A_j(u_i^{D*}/\alpha)} p_{i',j}$$
$$= \frac{2(1+\epsilon)}{(1-\beta)\epsilon} \sum_{i\in A_j(0)} p_{i,j} \sum_{i'\in A^*_{2,j}(0), u_{i'}^{D*}/\alpha \leq u_i^D} (u_{i'}^{D*} - u_{i'+1}^{D*})$$
$$\leq \frac{2(1+\epsilon)}{(1-\beta)\epsilon} \sum_{i\in A_j(0)} p_{i,j} u_i^D \alpha = \frac{2\alpha(1+\epsilon)}{(1-\beta)\epsilon} \sum_{i\in A_j(0)} w_i^D.$$

Since $\cup_{j\in\mathcal{S}} A^*_{2,j}(0) = A^*_2$ and $\cup_{j\in\mathcal{S}} A_j(0) = \mathcal{J}$, we conclude that $\sum_{i\in A^*_2} w_i^{D*} \leq \frac{2\alpha(1+\epsilon)}{(1-\beta)\epsilon} \sum_{i\in\mathcal{J}} w_i^D$. □

Based on the above lemmas, we conclude Theorem 2.

Theorem 2: For any $\epsilon \in (0, \frac{1}{2})$, O4A is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive.

Proof: By using lemma 1 and 3, we have

$$OPT^D = \sum_{i\in A_1^*\cup A_2^*} w_i^{D*} \leq \sum_{i\in A_1} w_i^D + \frac{2\alpha(1+\epsilon)}{(1-\beta)\epsilon} \sum_{i\in\mathcal{J}} w_i^D$$
$$\leq (1 + \frac{2\alpha(1+\epsilon)}{(1-\beta)\epsilon}) \sum_{i\in\mathcal{J}} w_i^D \leq \frac{1 + \frac{2\alpha(1+\epsilon)}{(1-\beta)\epsilon}}{1 - \frac{1+\beta\epsilon}{\beta\epsilon(\alpha-1)}} ALG.$$

Since all the utilities decrease by at most $\frac{1}{2}$ of the original after discretization, we have $OPT^D \geq \frac{1}{2} OPT$. Using $\alpha = 2 + 2/\epsilon$ and $\beta \in (\frac{1}{\epsilon(\alpha-2)}, 1)$ we deduce $OPT \leq O\left(\frac{1}{\epsilon^2}\right) ALG$. □

So far, we have proved a lower bound $\Omega(\frac{1}{\sqrt{\epsilon}})$ and an upper bound $O(\frac{1}{\epsilon^2})$ for the online job dispatching and scheduling problem where jobs with heterogeneous utility functions co-exist. In the proof of the lower bound, we only use a simple case where only two types of jobs with one single server are involved and the upload/download delay is not considered. Finding a tighter lower bound is promising if more complex cases (*e.g.*, considering multiple servers and upload/download delay) can be constructed. Nonetheless, this simple case can somewhat reflect the nature of this problem, that is, our lower bound justifies that the polynomial dependence on $\frac{1}{\epsilon}$ in our ratio is not avoidable, and this particularly means one cannot improve this ratio significantly to, *e.g.*, $O(\log \frac{1}{\epsilon})$.

F. Distributed Implementation

The original version of O4A appears to be centralized. However, since edge servers are usually scattered in different geographic locations in real-world scenarios, a centralized controller might be impractical for scheduling latency-sensitive jobs even with remote clouds for coordinated management. A trivially designed distributed algorithm may result in a large amount of information exchange. Therefore, it requires a special design to decide between which nodes the communication takes place to keep the original performance as much as possible with low information exchange.

Based on the attributes of O4A, we devise an implementation in the distributed environment via the following 3 steps: 1) When job i is released at time r_i, it sends request messages with its job information to each possible server $j \in \mathcal{S}$, except the remote cloud server; 2) When server j receives the request message from job i, it calculates the tentative finish time of job i, using Algorithm 1; and 3) Job i picks the server with the earliest tentative finish time for dispatching.

Algorithm 3 Distributed O4A

1 **Input** *job i, maximum number of connected servers k*;
2 Tentative finish time if job i is dispatched to cloud $f_{i,\text{cloud}} = \Delta^{\uparrow}_{i,\text{cloud}} + p_{i,\text{cloud}} + \Delta^{\downarrow}_{i,\text{cloud}}$;
3 Let $\mathcal{S}_{i,k}$ be the set of k edge servers with smallest $\Delta^{\uparrow}_{i,j} + p_{i,j} + \Delta^{\downarrow}_{i,j}$;
4 **for** $j \in \mathcal{S}_{i,k}$ **do**
5 Send request messages with its job information to server j;
6 When server j receives the request message from job i, it calculates the tentative finish time of job i using Algorithm 1 and replies $f_{i,j}$ to the mobile device;
7 Job i is dispatched to server $j = \arg\min_{j'\in\mathcal{S}_{i,k}\cup\{\text{cloud}\}} \{f_{i,j'}\}$;

Theoretically, this distributed method can achieve the same competitive ratio as the centralized version if the latency of querying a server's status can be ignored; otherwise, any algorithm that relies on the status of the server does not have a bounded ratio, even under the speed augmentation model, *i.e.*, we can always construct some hard instance based on the querying latency. However, simply using the above distributed method will incur $\Theta(|\mathcal{S}|)$ message complexity for each request and will cause serious abuse of network resources, where $|\mathcal{S}|$ is the number of edge servers in the system. Besides, the dispatching decision is made after all the responses return to the mobile device, which also slows down the dispatching process, especially when the number of servers is large.

To make the distributed implementation practical, we take advantage of the power-of-k choices [18]. That is, instead of querying all servers, each request can only communicate with k edge servers. For each job i, DO4A will pick at most k edge servers with the minimum possible finish time. Here, the minimum possible finish time for job i at server j is calculated as $\Delta^{\uparrow}_{i,j} + p_{i,j} + \Delta^{\downarrow}_{i,j}$. Furthermore, we avoid obtaining the tentative finish time on the remote cloud server by communicating with the cloud since it suffers from long communication latency. Instead, as the remote cloud has abundant resources compared with the edge, we choose to approximately estimate its tentative finish time locally by assuming there is no queuing delay but only upload/download delay and processing time.

The detail of distributed O4A (*i.e.*, DO4A) is shown in Algorithm 3. First, the mobile device calculates the tentative finish time, $f_{i,\text{cloud}}$, if job i is dispatched to the remote cloud (Line 2). Then, the mobile device selects k edge servers with the smallest $\Delta^{\uparrow}_{i,j} + p_{i,j} + \Delta^{\downarrow}_{i,j}$ as candidate servers (Line 3). Afterward, the device sends the information of job i to each candidate edge server $j \in \mathcal{S}_{i,k}$, including the upload/download delay, the processing time, and the utility function (Line 5). Each server calculates the tentative finish time of job i, *i.e.*, $f_{i,j}$, using Algorithm 1 and replies it to the mobile device (Line 6). Finally, job i is dispatched to the server with the smallest $f_{i,j}$ or $f_{i,\text{cloud}}$ if the remote cloud is chosen (Line 7).

In Algorithm 3, the message complexity for each request can be $\Theta(k)$, where k is a small constant. Compared with the original method of querying all servers, the message

complexity is drastically reduced (from $\Theta(|\mathcal{S}|)$ to $\Theta(k)$). As we will illustrate in Sec. V, in most cases by setting $k = 10$, DO4A can achieve almost the same performance as O4A, even when there are totally hundreds of edge servers.

V. Performance Evaluation

We evaluate O4A and DO4A with production-trace driven testbed experiments and large-scale simulations. By default, we set $\epsilon = 0.02$, $\alpha = 2 + \frac{2}{\epsilon}$, $\beta = 0.8$ and $k = 5$. Overall, our key findings are:

- In the testbed experiments running deep learning inference jobs, O4A performs 21% and 60% better than OnDisc and Random, respectively, when there are multiple types of utility functions. Besides, the total utility obtained by O4A is close to the optimal with a gap no larger than 12%.
- When the job transmission time and processing time are misestimated, O4A is more robust compared with the baseline algorithms. In specific, the average utility is degraded by at most 5% under 20% estimation error.
- The performance loss of DO4A is only 2% compared with O4A, even when the number of connected servers $k = 5$.
- Extensive simulations show that both O4A and DO4A are robust to various workload and parameter variations and consistently perform better than state-of-the-art baselines.

A. Experimental Settings

1) Testbed Cluster: Our edge-cloud testbed consists of 20 workers: 10 Nvidia Jetson Nano's and 10 Raspberry Pi 4b's. A DELL Precision 7920 Workstation is used as the scheduler to manage the cluster. Each Nvidia Jetson Nano has 4 cores, 4 GB RAM, 16 GB storage and a Gigabit Ethernet port. Each Raspberry Pi 4b has 4 cores, 1 GB RAM, 32 GB storage and a Gigabit Ethernet port. The workstation has two Intel Xeon Gold 6230 processors (each with 20 cores), 256 GB RAM and a 4 TB SSD. We use MQTT v3.1.1 as the communication protocol, which is widely used by IoT and mobile devices.

2) Workloads: We choose deep learning inference jobs as our workloads to evaluate O4A and the baseline algorithms. Table I lists the deep learning models of the jobs and their performance on Nvidia Jetson Nano and Raspberry Pi 4b. Specifically, each job requests to classify an image from ImageNet [19] using one of the models listed in Table I. The release time of jobs is set based on the trace of Google's production clusters [20]. In the testbed experiments, the average job release rate is $\sim$15 jobs per second. Note that since Nvidia Jetson Nano and Raspberry Pi 4b have different hardware specifications and run different platform-optimized inference engines, a job's processing time can be different on the two types of devices. This practical scenario fits in the unrelated machine system model that allows heterogeneous job processing time on different edge servers.

3) Utility Functions: To express jobs' heterogeneous sensitivity to JRT, each job is associated with a job-specific utility function when it is submitted. Similar to [27], we adopt the following functions in our experiments. When a job arrives, we randomly choose a utility function in the following list with equal probability (where parameters a, b, c below are selected uniformly at random in their respective intervals).

TABLE I
Inference Time of Popular ML Models

Model	Nvidia Jetson Nano (MXNet v1.8.0)	Raspberry Pi 4b (MXNet v1.8.0)
DenseNet-121 [21]	245 ± 60ms	141 ± 2ms
DenseNet-169	317 ± 74ms	195 ± 6ms
MobileNet 0.25 [22]	84 ± 24ms	37 ± 10ms
MobileNet 0.5	81 ± 21ms	35 ± 1ms
MobileNetV2 0.25 [23]	125 ± 30ms	59 ± 1ms
MobileNetV2 0.5	124 ± 32ms	59 ± 1ms
ResNet-18 V1 [24]	71 ± 17ms	29 ± 1ms
ResNet-18 V2 [25]	70 ± 23ms	28 ± 1ms
SqueezeNet 1.0 [26]	73 ± 17ms	29 ± 1ms

- Linear Function: $g(t) = -at + b, a, b > 0$,
  - we set $a \in [10, 20], b \in [20, 100]$
- All or Nothing (AoN) Function: $g(t) = \begin{cases} a & t < c \\ 0 & t \geq c \end{cases}$
  - we set $a \in [1, 100], c \in [0.5, 5]$
- Sigmoid Function: $g(t) = \dfrac{b}{e^{a(t-c)} + 1}, a, b, c > 0$
  - we set $a \in [1, 10], b \in [50, 100], c \in [0.5, 5]$

Jobs with the linear utility are usually latency-sensitive whose utility diminishes with longer JRT. A job associated with a hard deadline can express its utility with an AoN function: obtaining a utility of a if finished within its deadline c, and 0 else. The sigmoid utility can be viewed as a mix of the linear and the AoN function, using a decay factor (*i.e.*, a) to express the latency-sensitivity.

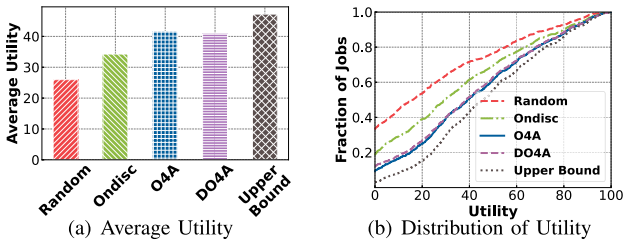
4) Upload/Download Delay: In theoretical analysis, we assume that the upload/download delays of all servers are given. In testbed experiments, the delays are estimated by testing Round-trip Time (RTT) per second. To flatten the jitter of the network, we update the delays using $\Delta_{\text{new}} = \gamma * \Delta_{\text{old}} + (1 - \gamma) * \text{RTT}_{\text{new}}/2$, where γ is set to 0.9 by default.

5) Baselines: We compare O4A with the following baselines:

- Random: dispatching jobs to servers randomly. On each server, the job with the highest potential utility is picked to process.
- OnDisc [4]: dispatching a job to the server which causes the least increase in job completion time. On each server, the scheduling discipline of the shortest remaining processing time-first is followed.
- Upper Bound: For any job i, its possible utility cannot exceed $\max_{j \in \mathcal{S}} g_i(p_{i,j} + \Delta^{\uparrow}_{i,j} + \Delta^{\downarrow}_{i,j})$, where only the inevitable processing and communication time are counted, whereas the possible queuing delay is ignored. Therefore, an upper bound of the aggregate utility for all jobs is $\sum_{i \in \mathcal{J}} \max_{j \in \mathcal{S}} g_i(p_{i,j} + \Delta^{\uparrow}_{i,j} + \Delta^{\downarrow}_{i,j}))$, which helps us assess how close each algorithm is to the optimal.

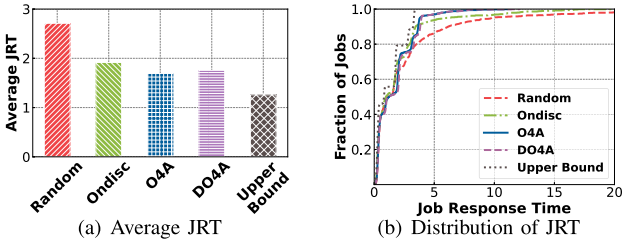
B. Testbed Experiment: Overall Comparison

1) Improvement on Utility: We first evaluate the overall performance of O4A and compare it with Random, OnDisc and Upper Bound. As mentioned above, the utility function for each job is selected uniformly from linear, sigmoid and



(a) Average Utility (b) Distribution of Utility

Fig. 4. Average utility and distribution of utility. (All utility functions.)

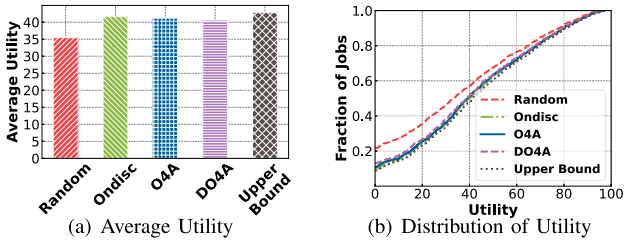


(a) Average JRT (b) Distribution of JRT

Fig. 5. Average JRT and distribution of JRT. (All utility functions.)

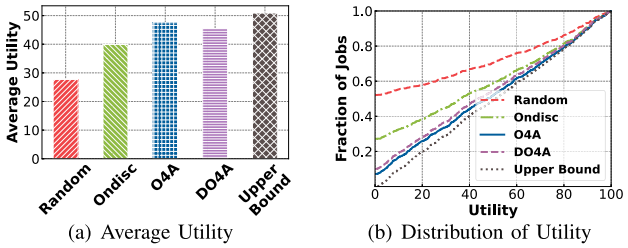
AoN functions. The experimental results are shown in Fig. 4, which illustrates the average utility of all jobs as well as the cumulative density function (CDF) for the utility each job gains at the time of its completion. As shown in Fig. 4(a), O4A can achieve about 21% and 60% higher utility than OnDisc and Random, respectively. Besides, the utility loss of DO4A compared with O4A is only 2%. Fig. 4(b) shows that compared with Random and OnDisc, O4A achieves higher utility on almost all percentiles. Furthermore, the gap between O4A and the optimal solution (upper-bounded using Upper Bound) is at most 12%, which reveals that the total utility obtained by O4A is close to the optimal.

2) Improvement on Job Response Time: Fig. 5 presents the overall JRT in the testbed experiment. Although OnDisc is designed for optimizing JRT, the average JRT of O4A's is still 10% lower than OnDisc's, *cf.* Fig. 5(a). Besides, the JRT of DO4A is 4% higher than O4A's. Inappropriate dispatching caused by misestimation may result in unbalanced server load and thus the server resources cannot be fully utilized, which causes OnDisc's degraded performance. O4A is more robust to misestimation since it assigns each job with a larger tentative interval, so that it can assure a higher probability of completion at the expense of (slightly) lowering the tentative utility. Fig. 5(b) shows the distribution of the job completion time. Observe that O4A performs slightly worse than OnDisc on short jobs, because OnDisc tends to favor more short jobs but can starve long ones. Overall, O4A achieves comparable or better performance compared with state-of-the-art JRT-optimized algorithms. The problem we studied in this paper aims to maximize overall utility. So, indeed there might be some jobs never get served when the load exceeds the computational resource of servers. In practice, jobs that miss their tentative finish time (or jobs out of their tentative intervals) will be set with a lower priority. When there are no jobs in their tentative intervals at some point, jobs with lower priority will be processed. And when the load of a server is high, *i.e.*, lots of jobs are queuing in this server, subsequent requests will be forwarded to other servers or the remote cloud to mitigate the load on this server. That is, jobs with lower utility will finally be served. The experimental results also show that there is almost no job with an extremely large response time. Furthermore, to achieve better fairness, a fairness knob can be added to the algorithm just like what has been done in [28].



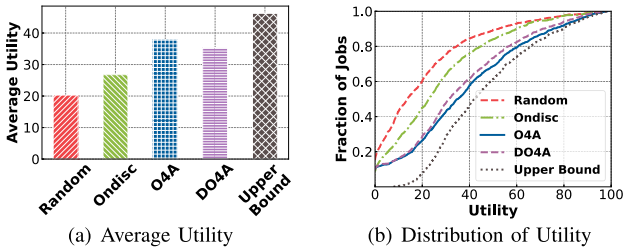
(a) Average Utility (b) Distribution of Utility

Fig. 6. Average utility and distribution of utility. (Linear utility only.)



(a) Average Utility (b) Distribution of Utility

Fig. 7. Average utility and distribution of utility. (AoN utility only.)



(a) Average Utility (b) Distribution of Utility

Fig. 8. Average utility and distribution of utility. (Sigmoid utility only.)

C. Testbed Experiment: Understanding Utility Functions

To study the impact of different types of utility functions, we conduct several experiments. In each experiment, we set a job's utility function to a specific type and measure the performance of all algorithms.

1) Impact of Linear Function: Fig. 6 shows the average utility of all algorithms and their distribution in the experiment that only submits jobs with linear utility functions. As we can see, the curves of OnDisc and O4A are very close to the CDF curve of Upper Bound. More specifically, OnDisc, O4A and DO4A can achieve about 17%, 16% and 14% more utility than Random, and gains 97%, 96% and 93% average utility of Upper Bound, respectively. This is because OnDisc performs the shortest job first strategy, which is actually achieving the same target as the linear utility function.

2) Impact of AoN Function: Fig. 7 shows average utility and utility CDF for all algorithms when only submitting jobs with AoN utilities. O4A achieves not only higher utility, but also a substantially lower deadline miss ratio (*i.e.*, the ratio of jobs with zero utility). In Fig.7(b), the percentage of jobs with non-zero utility is exactly the deadline miss ratio: O4A and DO4A only miss 7% and 10% job deadlines, respectively, while OnDisc and Random miss 27% and 52% jobs, respectively. Since both baseline algorithms are deadline-agnostic (they only seek to optimize job completion time), they

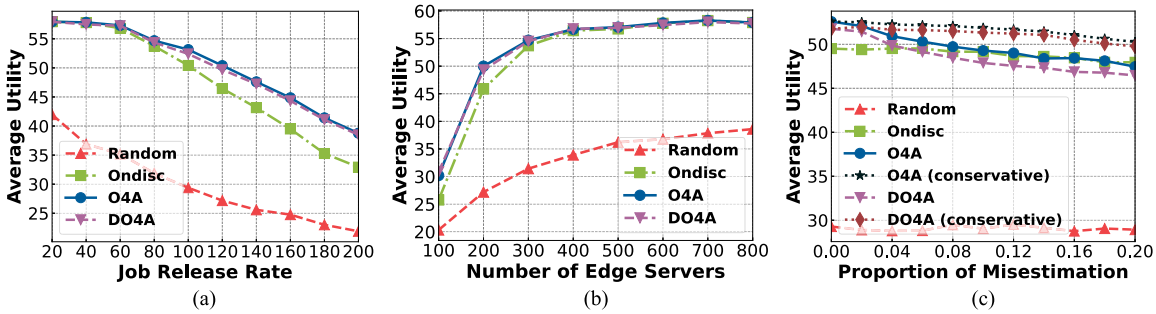


Fig. 9. Sensitivity analysis: (a) Impact of job release rate. (b) Impact of the number of servers. (c) Impact of the misestimation.

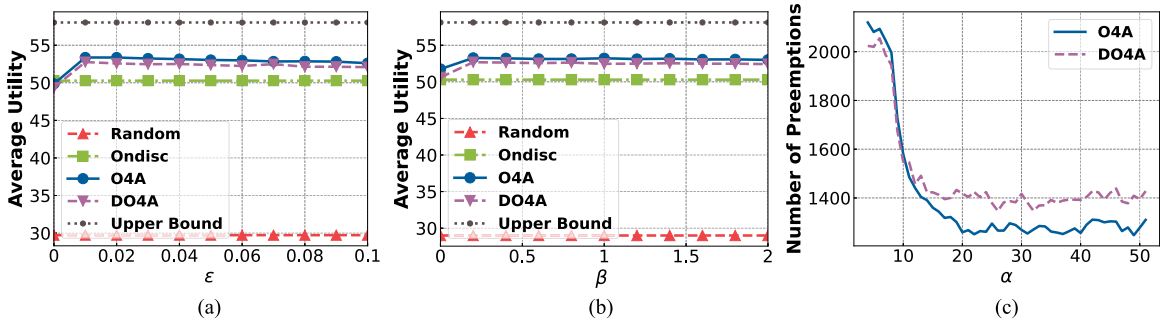


Fig. 10. Impact of some parameters. (a) Impact of parameter ϵ. (b) Impact of parameter β. (c) Impact of parameter α on the number of preemptions.

may waste workers' time in processing jobs whose deadline can hardly be satisfied.

3) Impact of Sigmoid Function: Fig. 8 shows the average utility and utility CDF for all algorithms when only submitting jobs with sigmoid utility functions. The two baseline algorithms perform much worse on sigmoid utility functions, compared with linear and AoN functions: O4A achieves 86% and 41% higher utility than Random and OnDisc, respectively. The degraded performance of the two baselines is mainly due to the non-linearity of the sigmoid utility function. It should be noted that the utility gained by DO4A is only 94% of O4A, and the reason for the relatively poor performance of DO4A under non-linear utility functions may require more detailed exploration.

D. Large-Scale Simulations

We conduct sensitivity studies by large-scale simulations in a cluster of 300 edge servers based on production traces. Specifically, we investigate the impact of the job release rate, the number of edge servers, the misestimation of job information, the speed augmentation parameter ϵ and α, and the number of connected servers of mobile devices.

1) Impact of Job Release Rate: We present the trend with rate changes in Fig. 9(a), by increasing the job release rate from 20 to 200. Due to limited resources, the average utility gradually decreases with the increase of job release rate, for all algorithms. However, the decrease in average utility that O4A experiences is slower than OnDisc's, *i.e.*, we deduce that O4A schedules jobs more efficiently when cluster load becomes heavier. Besides, no matter how the job release rate changes, the performance of DO4A is always kept near O4A's.

2) Impact of the Number of Edge Servers: Fig. 9(b) shows the trend of the average utility with a variable number of edge servers. All algorithms benefit from adding edge servers. Random only obtains a marginal increase on the average utility, while OnDisc, O4A and DO4A attain higher gains. Similar to the result of the impact of job release rate, the performance of DO4A is still closed to O4A's under any number of servers.

3) Impact of Inaccurate Estimation: In an actual production environment, it is hard to acquire an accurate estimation of the job processing time or the transmission delay. Fig. 9(c) investigates the impact of misestimation, where a scheduler knows the estimated job processing time $\hat{p}_{i,j}$, estimated upload delay $\hat{\Delta}^{\uparrow}_{i,j}$, estimated download delay $\hat{\Delta}^{\downarrow}_{i,j}$ and the estimated error bound δ. The true values of $p_{i,j}, \Delta^{\uparrow}_{i,j}, \Delta^{\downarrow}_{i,j}$ are in the range of $[(1-\delta)\hat{p}_{i,j}, (1+\delta)\hat{p}_{i,j}]$, $[(1-\delta)\hat{\Delta}^{\uparrow}_{i,j}, (1+\delta)\hat{\Delta}^{\uparrow}_{i,j}]$, $[(1-\delta)\hat{\Delta}^{\downarrow}_{i,j}, (1+\delta)\hat{\Delta}^{\downarrow}_{i,j}]$, respectively. We vary the value of δ from 0 to 0.2. When O4A undertakes scheduling decisions based on $\hat{p}_{i,j}, \hat{\Delta}^{\uparrow}_{i,j}, \hat{\Delta}^{\downarrow}_{i,j}$, it is heavily impacted by increasing estimation errors. To obtain a more robust method, we apply O4A and DO4A using a conservative estimate of job processing time and transmission delays, *i.e.*, $(1+\delta)\hat{p}_{i,j}$,$(1+\delta)\hat{\Delta}^{\uparrow}_{i,j}$,$(1+\delta)\hat{\Delta}^{\uparrow}_{i,j}$. The simulation shows that this improvement indeed renders O4A and DO4A more robust to estimation errors, even when the misestimation is up to 20%.

4) Impact of the Speed Augmentation Parameters: In Fig. 10(a), we study the impact of varying the speed augmentation parameter ϵ from 0 to 0.1. We unravel that altering the value of ϵ has little impact on the performance of O4A and DO4A. Recall that O4A's parameter $\alpha = 2+\frac{2}{\epsilon}$, tends to infinity as $\epsilon \to 0$, which corresponds to never preempting any job. This motivates the effect of choosing $\epsilon > 0$ so as to prevent O4A from becoming a non-preemptive scheduler. In Fig. 10(b), we look into the impact of parameter β, which determines the length of extra time of tentative intervals. Note that in theoretical analysis, the range of β is $(\frac{1}{\epsilon(\alpha-2)}, 1)$. Here, we derive from the experiment that the performance of O4A is

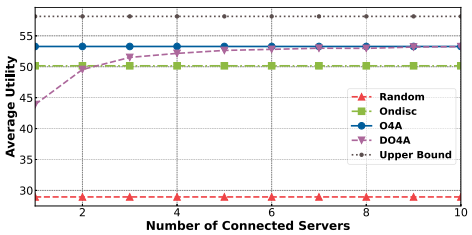


Fig. 11. Impact of the number of connected servers.

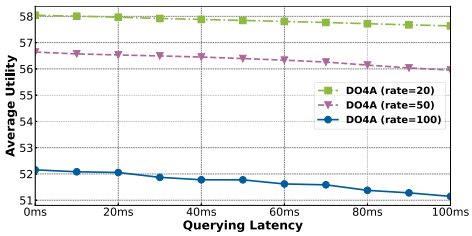


Fig. 12. Impact of query latency.

nearly insensitive to the value of β even when β is out of the theoretical valid interval. The reason is that in practice appropriate values of β can yield roughly accurate utility estimates and thus relatively better results. However, setting $\beta = 0$ will result in a poor performance, which indicates the importance of the existence of extra time. We also investigate the change of the number of preemptions as parameter α increases, *cf.* Fig. 10(c). As we can see, the decline rate of the number of preemptions starts to slow down when α is greater than 20. Intuitively, the number of preemptions will finally drop to 0 as $\alpha \to \infty$. From this perspective, when the penalty of preemption cannot be ignored, the value of α should be set to a number greater than 20 (corresponding to $\epsilon < 0.05$) in order to reduce the preemption penalty. We also found that with the increase of α, the preemption frequency of DO4A decreases more slowly than that of O4A, as the server candidates in DO4A are limited to a maximum number of k.

5) Impact of k in the Distributed Implementation: In the design of DO4A, we restrict the number of servers that each mobile device can communicate with to a constant k to reduce the message complexity. In Fig. 11 we show the impact if we change the parameter k. As we can see, when we set $k = 1$, *i.e.*, the case that all mobile devices only selfishly send requests to the server with the shortest possible finish time, the performance of DO4A is much worse than OnDisc. The performance of DO4A approaches close to the performance of O4A when $k = 5$ and almost the same with O4A's performance when $k \geq 10$, where the total number of edge servers in the system can be as large as 300.

6) Impact of Querying Latency: When the querying latency of DO4A is 0, DO4A has the same competitive ratio as O4A. Here, we show the tendency when the querying latency gets larger (Fig. 12), where the job release rates are set to 20, 50 and 100 per second, respectively. When the querying latency is less than 50ms, the performance loss of DO4A is small. And when the querying latency is greater than 50ms, the magnitude of the performance degradation with increasing querying latency begins to increase. This phenomenon is more obvious when the load is heavy, which may cause more jobs to miss their deadlines. Despite this, the performance loss of DO4A is less than 2% even when the query latency is 100ms.

VI. Discussion

A. Infrequent Job Switching

For any preemptive scheduling algorithm, job switching incurs a certain degree of switching overhead according to its execution environment (the overhead due to CPU context switching). For this reason, it is imperative that the scheduling algorithm avoid frequent switching when the switching overhead is high. Recall that O4A uses a preemption knob α to control the preemption threshold, which means that a job can preempt other jobs only when its density is high enough. Thus, by properly choosing the preemption knob α, O4A can adjust its switching frequency taking into consideration the switching overhead.

B. Robustness to Misestimation

The processing time and transmission delay of jobs in a production environment cannot be perfectly estimated; any scheduling algorithm that relies on accurate knowledge of the job processing/transmission time could make inefficient decisions when agnostic to the estimation error. Therefore, it is instrumental that the scheduling algorithm is aware of and robust to estimation errors. In Sec. V-D.3, we evaluate the impact of estimation error and show that a simple modification to O4A greatly improves its robustness. It is interesting to theoretically study the impact of estimation error on O4A's competitive ratio, which will be a focal point for future research.

C. Comparison With Utility-Specific Algorithms

O4A is designed to maximize the utility for general heterogeneous utility functions (*i.e.*, heterogeneous jobs co-exist and each job may choose a different type of utility function). Scheduling jobs with a specific utility function is also an interesting topic to study. To the best of our knowledge, scheduling deadline-aware jobs [16], [29]–[32] is the most common setting besides general utility functions. In the deadline-aware model, a fixed utility can be earned if a job is finished before its deadline, and otherwise, the utility is zero, called All-or-Nothing (AoN) utility function. To maximize the total AoN utility, Sanjoy *et al.* [16], [29], [30] proposed an $O(K)$-competitive deterministic algorithm, where K is the ratio of the maximum to the minimum job density. The algorithm is later improved to be a randomized $O(\min\{\log K, \log \psi\})$-competitive algorithm by Gilad *et al.* [31] and Bala *et al.* [32], where ψ is the ratio of the maximum to the minimum job size. All the above ratios are derived without speed augmentation. Specifically, our evaluation based on production traces reveals that O4A can achieve comparable performance to these algorithms designed for AoN utility on scheduling deadline-aware jobs. So far, there is no study related to scheduling jobs with the same type of utility function under the speed augmentation model. We propose this as a future research direction. And some previous results concerning response time minimization as OnDisc [4] may shed some light on this problem.

TABLE II
RELATED WORKS ON ONLINE SCHEDULING

Algorithm (Author)	Machine Model	Transmission Delay	Objective	Lower Bound	Upper Bound
Leonardi *et al.* [33]	Identical	✘	Minimize Weighted JRT	-	$O(\log(\min(n/m, P)))$
Garg *et al.* [34]	Related	✘	Minimize Weighted JRT	-	$O(\log^2 P)$
Anand *et al.* [35]	Unrelated	✘	Minimize Weighted JRT	Unbounded	$O(1/\epsilon)$
OnDisc [4], [28]	Unrelated	✔	Minimize Weighted JRT	-	$O(1/\epsilon)$
Bansal *et al.* [36]	Single	✘	Minimize Polynomial Cost	-	$O(1/\epsilon^k)$
Im *et al.* [37]	Single	✘	Minimize General Cost	$O(1)$-speed, $O(1)$	$(2+\epsilon)$-speed, $O(1)$
Gilad *et al.* [31], Bala *et al.* [32]	Single	✘	Maximize AoN Utility	$\Omega(\min\{\log K, \log \psi\})$	$O(\min\{\log K, \log \psi\})$
Krik *et al.* [38]	Identical	✘	Maximize General Utility	Unbounded (deterministic)	$O(1/\epsilon^3)$
Im *et al.* [17]	Unrelated	✘	Maximize General Utility	-	$O(1/\epsilon^2)$
O4A [this work]	Unrelated	✔	Maximize General Utility	$\Omega(1/\sqrt{\epsilon})$ (deterministic)	$O(1/\epsilon^2)$

Note: n and m are the number of jobs and servers. P is the ratio of the largest to the smallest job processing time. K is the ratio of the maximum to the minimum job density. ψ is the ratio of the maximum to the minimum job size. ϵ is the speed augmentation parameter.

VII. RELATED WORK

A. Classic Online Scheduling

Designing online scheduling algorithms for utility maximization has been studied for decades. One typical setting is that each job arrives online with a utility and a deadline. In the case of a single server and preemption allowed, Sanjoy *et al.* [16], [29], [30] conducted a series of influential works, where they resolved the problem complexity and figured out the optimal deterministic online algorithm. However, the competitive ratio depends on the instance of the set of jobs and can be extremely large. Gilad *et al.* [31] and Bala *et al.* [32] employed the power of randomization and the randomized competitive ratio is still related to the input instance. When scheduling on m servers, Samin *et al.* [39] presented a lower bound $c(\epsilon, m)$ of the competitive ratio of any deterministic online algorithm for the problem with $\epsilon \in (0, 1]$ if the preemption is not allowed. Due to this lower bound, some works [17], [38], [40], [41] resorted to the resource augmentation analysis, which means the servers running their algorithms can be $(1+\epsilon)$ faster than those running the optimal solution. In this line of work, Krik *et al.* [38] constructed a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^3})$ competitive algorithm for any fixed constant $\epsilon > 0$ under the identical severs setting. Considering the complexity of jobs, Agrawal [42] studied a setting where each job can be represented as a Directed Acyclic Graph (DAG) and proposed a $O(1)$-competitive algorithm under speed augmentation analysis. Bao *et al.* [43] studied online job scheduling in one ML cluster to maximize the overall utility of all jobs. A recent result from Im *et al.* [17] revealed that there is a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$ online algorithm in the unrelated server setting. We leverage similar ideas but consider a more general model with job upload and download delay. Moreover, we here derive a lower bound of the competitive ratio. In O4A, we further introduce techniques of logarithmic discretization, segment tree optimization, and timeline scrolling to achieve a low time complexity. To the best of our knowledge, O4A is the first practical algorithm to schedule jobs with heterogeneous utilities in edge computing.

Besides utility maximization, cost minimization is also an important objective in online scheduling. When all cost functions are linear, the problem is also known as weighted flow time minimization scheduling, which has been well studied. Leonardi and Raz [33] initiated the study on identical servers and showed that Shortest Remaining Processing Time (SRPT) algorithm has a competitive ratio of $O(\log(\min(\frac{n}{m}, P)))$, where n, m and P are the number of jobs, the number of servers and the ratio of largest processing time to the smallest processing time, respectively. Azar *et al.* [44] considered the problem of online scheduling on a single machine in order to minimize weighted flow time with uncertain processing time and the proposed algorithms match the best-known competitiveness bounds. Han *et al.* [45] investigated gang scheduling with job placement, where all the workers must be allocated and scheduled synchronously. A series of works [46], [47] then improved the algorithm by providing more practical models while preserving the competitive ratio. Garg and Kumar [34] were the first to extend the problem on related servers, a setting where each server i has a speed s_i. Still considering weighted flow time minimization, they showed an algorithm with competitive ratio $O(\log^2 P)$. However, the same authors [8] later showed no competitive algorithm is possible for unrelated servers even considering linear cost functions only. Thus, Garg *et al.* [9] resorted to speed augmentation analysis and figured out a $(1+\epsilon)$-speed $O((1+\epsilon^{-1})^2)$-competitive algorithm for weighted flow time minimization. Unlike the exploration in the field of weighted flow time minimization, researchers made slow progress on the general cost function. Bansal and Pruhs [36] studied this problem with polynomial cost functions on a single server and showed a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^k})$-competitive algorithm, where k is the degree of the polynomials. Im *et al.* [37] extended the result by considering cost minimization on a single server with non-decreasing functions. We categorize the above works in Table II.

B. Online Scheduling in Edge-Cloud Systems

Since resources in edge servers are relatively limited, extensive studies focus on resource management and load balancing in edge computing. With the practical assumption that user locations might keep changing, Urgaonkar *et al.* [48] modeled the workload scheduling problem as Markov Decision Process and adopted Lyapunov optimization to solve the problem. Tong *et al.* [49] designed a tree hierarchical edge-cloud and proposed an efficient heuristic workload dispatching policy to offload the workload. More recently, Meng *et al.* [7] took the network bandwidth into consideration and proposed

a deadline-aware job dispatching strategy. Zhou *et al.* [50] considered a penalty function that characterizes the cost of violating the soft deadlines for cloud resource provisioning. Wang *et al.* [51] formulated the dynamic resource configuration as a multi-period online cost minimization problem, and proposed an online mean field aided resource configuration policy. All the aforementioned works rely on stochastic optimization and assume that the job releasing pattern follows some specific distribution, which might not hold in practice. Instead, Tan *et al.* [4] investigated online job dispatching and scheduling in edge computing, where jobs arrive at arbitrary time and order. They proposed an efficient online algorithm with a competitive ratio of $O(\frac{1}{\epsilon})$ under the $(1+\epsilon)$-speed augmentation model. Liu *et al.* [52] studied task placement and scheduling when there are dependencies between tasks. As stated above, previous works mainly focused on a single specific metric. However, in practice, a variety of applications can co-exist, which employ different optimizing metrics, *i.e.*, to indicate their different levels of sensitivity to latency. Therefore, to capture realistic scenarios in edge computing, in this work, we investigate jobs of heterogeneous utilities, where the utility can be any non-increasing function of the JRT, not necessarily linear or convex, and more importantly jobs with heterogeneous utilities functions are allowed to co-exist in the system.

VIII. Conclusion

Motivated by real applications, we studied online dispatching and scheduling of jobs with heterogeneous utility functions in edge computing. We have obtained a lower bound $\Omega(\frac{1}{\sqrt{\epsilon}})$ and an upper bound $O(\frac{1}{\epsilon^2})$ for this problem, both under the $(1+\epsilon)$-speed augmentation model. We also extended our algorithm O4A to its distributed version, *i.e.*, DO4A, which reduces the communication complexity while maintaining the original characteristics. Our testbed experiments and extensive production trace-driven simulations illustrate that O4A can increase the total utility by up to 50% compared with state-of-the-art baselines, while also achieving comparable or better average job response time and deadline miss ratio. And the performance loss of DO4A is only 2% compared with O4A. Besides maximizing the aggregate utility of all jobs, an interesting point of future work remains to incorporate fairness in job scheduling. Besides, in light of the fact that acquiring accurate information on job processing time is difficult (due to the unpredictable contention on computation and communication resources), another meaningful extension is to theoretically analyze the impact of estimation error on the performance of the dispatching and scheduling strategies.

References

[1] C. Zhang *et al.*, "Online dispatching and scheduling of jobs with heterogeneous utilities in edge computing," in *Proc. 21st Int. Symp. Theory, Algorithmic Found., Protocol Design Mobile Netw. Mobile Comput.*, Oct. 2020, pp. 101–110.

[2] L. Liu *et al.*, "Cutting the cord: Designing a high-quality untethered VR system with low latency remote rendering," in *Proc. Mobicom*, 2018, pp. 68–80.

[3] J. Feng, Z. Liu, C. Wu, and Y. Ji, "AVE: Autonomous vehicular edge computing framework with ACO-based scheduling," *IEEE Trans. Veh. Technol.*, vol. 66, no. 12, pp. 10660–10675, Dec. 2017.

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

[5] H. Shah-Mansouri and V. W. S. Wong, "Hierarchical fog-cloud computing for IoT systems: A computation offloading game," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 3246–3257, Aug. 2018.

[6] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, "Service entity placement for social virtual reality applications in edge computing," in *Proc. INFOCOM*, Apr. 2018, pp. 468–476.

[7] J. Meng, H. Tan, X.-Y. Li, Z. Han, and B. Li, "Online deadline-aware task dispatching and scheduling in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1270–1286, Jun. 2019.

[8] N. Garg and A. Kumar, "Minimizing average flow-time: Upper and lower bounds," in *Proc. 48th Annu. IEEE Symp. Found. Comput. Sci. (FOCS)*, Oct. 2007, pp. 603–613.

[9] J. S. Chadha, N. Garg, A. Kumar, and V. Muralidhara, "A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation," in *Proc. STOC*, 2009, pp. 679–684.

[10] S.-C. Lin *et al.*, "The architectural implications of autonomous driving: Constraints and acceleration," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 751–766, 2018.

[11] F. Vicente, Z. Huang, X. Xiong, F. D. L. Torre, W. Zhang, and D. Levi, "Driver gaze tracking and eyes off the road detection system," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 4, pp. 2014–2027, Aug. 2015.

[12] D. Sadigh *et al.*, "Data-driven probabilistic modeling and verification of human driver behavior," in *Proc. AAAI Spring Symp.*, 2014, pp. 1–8.

[13] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2174–2182.

[14] X. Zhang, S. Sen, D. Kurniawan, H. Gunawi, and J. Jiang, "E2E: Embracing user heterogeneity to improve quality of experience on the web," in *Proc. ACM SIGCOMM*, 2019, pp. 289–302.

[15] R. Miller. (2020). *Rolling Zettabytes: Quantifying Data Impact Connected Cars*. [Online]. Available: https://datacenterfrontier.com/rolling-zettabytes-quantifying-the-data-impact-of-connected-cars/

[16] S. Baruah *et al.*, "On the competitiveness of on-line real-time task scheduling," *Real-Time Syst.*, vol. 4, no. 2, pp. 125–144, Jun. 1992.

[17] S. Im and B. Moseley, "General profit scheduling and the power of migration on heterogeneous machines," in *Proc. 28th ACM Symp. Parallelism Algorithms Architectures*, Jul. 2016, pp. 165–173.

[18] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1094–1104, Oct. 2001.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. CVPR*, Jun. 2009, pp. 248–255.

[20] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format + schemam," Google, Mountain View, CA, USA, Tech. Rep., Nov. 2011. [Online]. Available: https://github.com/google/cluster-data

[21] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.

[22] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[23] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, Jun. 2016, pp. 770–778.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. ECCV*, 2016, pp. 630–645.

[26] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50X fewer parameters and $\leq$0.5 MB model size," 2016, *arXiv:1602.07360*.

[27] L. Chen, W. Cui, B. Li, and B. Li, "Optimizing coflow completion times with utility max-min fairness," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[28] Z. Han, H. Tan, X.-Y. Li, S. H.-C. Jiang, Y. Li, and F. C. M. Lau, "OnDisc: Online latency-sensitive job dispatching and scheduling in heterogeneous edge-clouds," *IEEE/ACM Trans. Netw.*, vol. 27, no. 6, pp. 2472–2485, Dec. 2019.

[29] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha, "On-line scheduling in the presence of overload," in *Proc. 32nd Annu. Symp. Found. Comput. Sci.*, 1991, pp. 100–110.

[30] G. Koren and D. Shasha, "Dover: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems," *SIAM J. Comput.*, vol. 24, no. 2, pp. 318–339, 1995.
[31] G. Koren and D. Shasha, "MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling," *Theor. Comput. Sci.*, vol. 128, nos. 1–2, pp. 75–97, Jun. 1994.
[32] B. Kalyanasundaram and K. Pruhs, "Fault-tolerant real-time scheduling," *Algorithmica*, vol. 28, no. 1, pp. 125–144, Sep. 2000.
[33] S. Leonardi and D. Raz, "Approximating total flow time on parallel machines," in *Proc. STOC*, 1997, pp. 1–10.
[34] N. Garg and A. Kumar, "Minimizing average flow time on related machines," in *Proc. 38th Annu. ACM Symp. Theory Comput. (STOC)*, 2006, pp. 730–738.
[35] S. Anand, N. Garg, and A. Kumar, "Resource augmentation for weighted flow-time explained by dual fitting," in *Proc. 23rd Annu. ACM-SIAM Symp. Discrete Algorithms*, Jan. 2012, pp. 1228–1241.
[36] N. Bansal and K. R. Pruhs, "Server scheduling to balance priorities, fairness, and average quality of service," *SIAM J. Comput.*, vol. 39, no. 7, pp. 3311–3335, Jan. 2010.
[37] S. Im, B. Moseley, and K. Pruhs, "Online scheduling with general cost functions," *SIAM J. Comput.*, vol. 43, no. 1, pp. 126–143, Jan. 2014.
[38] K. Pruhs and C. Stein, "How to schedule when you have to buy your energy," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Berlin, Germany: Springer, 2010, pp. 352–365.
[39] S. Jamalabadi, C. Schwiegelshohn, and U. Schwiegelshohn, "Commitment and slack for online load maximization," in *Proc. 32nd ACM Symp. Parallelism Algorithms Architectures*, Jul. 2020, pp. 339–348.
[40] N. Bansal, H.-L. Chan, and K. Pruhs, "Competitive algorithms for due date scheduling," *Algorithmica*, vol. 59, no. 4, pp. 569–582, Apr. 2011.
[41] L. Chen, F. Eberle, N. Megow, K. Schewior, and C. Stein, "A general framework for handling commitment in online throughput maximization," in *Proc. IPCO*, 2019, pp. 1–28.
[42] K. Agrawal, J. Li, K. Lu, and B. Moseley, "Scheduling parallelizable jobs online to maximize throughput," in *Proc. Latin Amer. Symp. Theor. Informat.* Cham, Switzerland: Springer, 2018, pp. 755–776.
[43] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 495–503.
[44] Y. Azar, S. Leonardi, and N. Touitou, "Flow time scheduling with uncertain processing time," in *Proc. 53rd Annu. ACM SIGACT Symp. Theory Comput.*, Jun. 2021, pp. 1070–1080.
[45] Z. Han *et al.*, "SPIN: BSP job scheduling with placement-sensitive execution," *IEEE/ACM Trans. Netw.*, vol. 29, no. 5, pp. 2267–2280, Oct. 2021.
[46] C. Chekuri, S. Khanna, and A. Zhu, "Algorithms for minimizing weighted flow time," in *Proc. STOC*, 2001, pp. 84–93.
[47] N. Avrahami and Y. Azar, "Minimizing total flow time and total completion time with immediate dispatching," *Algorithmica*, vol. 47, no. 3, pp. 253–268, Mar. 2007.
[48] R. Urgaonkar *et al.*, "Dynamic service migration and workload scheduling in edge-clouds," *Perform. Eval.*, vol. 91, pp. 205–228, Sep. 2015.
[49] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.
[50] R. Zhou, Z. Li, C. Wu, and Z. Huang, "An efficient cloud market mechanism for computing jobs with soft deadlines," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 793–805, Apr. 2016.
[51] Z. Wang, J. Ye, and J. C. Lui, "An online mean field approach for hybrid edge server provision," in *Proc. MobiHoc*, 2021, pp. 131–140.
[52] L. Liu, H. Tan, S. H.-C. Jiang, Z. Han, X.-Y. Li, and H. Huang, "Dependent task placement and scheduling with function configuration in edge computing," in *Proc. Int. Symp. Quality Service*, Jun. 2019, pp. 1–10.

Chi Zhang received the B.Eng. degree (Hons.) in computer science and technology from the University of Science and Technology of China (USTC) under the Talent Program in computer and information science and technology in 2017, where he is currently pursuing the Ph.D. degree with the School of Computer Science and Technology. His main research interests are data center networking, cloud computing, and algorithms.

Haisheng Tan (Senior Member, IEEE) received the B.E. degree (Hons.) in software engineering and the B.S. degree (Hons.) in management from the University of Science and Technology of China (USTC), and the Ph.D. degree in computer science from the University of Hong Kong (HKU). He is currently a Professor at USTC. He has published over 80 papers in prestigious journals and conferences, mainly in the areas of AIoT and edge computing. His research interests include algorithms and networking. He recently received the Best Paper Award in WASA'19, CWSN'20, PDCAT'20, and ICAPDS'21.

Haoqiang Huang received the B.Eng. degree in computer science and technology from the University of Science and Technology of China in 2019. He is currently pursuing the Ph.D. degree in computer science and engineering with the Hong Kong University of Science and Technology. His research interests include online algorithm and algorithm on massive data set.

Zhenhua Han received the B.Eng. degree in electronic and information engineering from the University of Electronic Science and Technology of China in 2014 and the Ph.D. degree from the University of Hong Kong (HKU). He currently works as a Researcher at Microsoft Research (Asia). His research interests include cloud computing, cluster scheduling, machine learning systems, online algorithms, and stochastic optimization. Many of his works have been published in top venues, such as USENIX OSDI, IEEE INFOCOM, and IEEE/ACM ToN.

Shaofeng H.-C. Jiang received the Ph.D. degree from the University of Hong Kong. Before, he joined PKU, he worked as a Post-Doctoral Researcher with the Weizmann Institute of Science and then as an Assistant Professor with Aalto University. He is currently an Assistant Professor with the Center on Frontiers of Computing, Peking University. His research interests are generally theoretical computer science, with a focus on algorithms for massive datasets, online algorithms, and approximation algorithms.

Guopeng Li received the B.Eng. degree in computer science and technology from Central South University in 2020. He is currently pursuing the Ph.D. degree with the University of Science and Technology of China (USTC). His main research interests are cloud computing, edge computing, and algorithm design.

Xiang-Yang Li (Fellow, IEEE) received the bachelor's degree from the Department of Computer Science in 1995, the bachelor's degree from the Department of Business Management, Tsinghua University, in 1995, and the M.S. and Ph.D. degrees from the Department of Computer Science, University of Illinois at Urbana–Champaign, in 2000 and 2001, respectively. He was a Full Professor with the Illinois Institute of Technology, Chicago, USA. He is currently a Full Professor and the Executive Dean of the School of Computer Science and Technology, USTC, Hefei, China. He is an ACM Fellow and an ACM Distinguished Scientist. He published a monograph *Wireless Ad Hoc and Sensor Networks: Theory and Applications*. His research span artificial intelligent Internet of Things, mobile computing, data sharing and trading, and privacy.