# Online Dispatching and Scheduling of Jobs with Heterogeneous Utilities in Edge Computing

Chi Zhang[1], Haisheng Tan[1], Haoqiang Huang[1], Zhenhua Han[2], Shaofeng H.-C. Jiang[3], Nikolaos Freris[1], Xiang-Yang Li[1]
[1] LINKE Lab, University of Science and Technology of China, Hefei, China
[2] Microsoft Research Asia, Shanghai, China
[3] Weizmann Institute of Science, Israel

## 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 works in edge computing mainly focused on either one kind of utility function (*e.g.*, linear, sigmoid, or the hard deadline) or different kinds of utilities separately. In this paper, we investigate online job dispatching and scheduling strategies under the setting of coexistence of heterogeneous utilities, *i.e.*, various coexisting jobs can employ different non-increasing utility functions. The goal is to maximize the total utility over all jobs in an edge system. Besides heterogeneous utilities, we here adopt a practical online model where the unrelated machine model and the upload and download delay are considered. We proceed to propose an online algorithm, 04A, to dispatch and schedule jobs with heterogeneous utilities. Our theoretical analysis shows that 04A is $O(\frac{1}{\epsilon^2})$-competitive under the $(1+\epsilon)$-speed augmentation model, where $\epsilon$ is a small positive constant. We implement 04A 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 04A can increase the total utility by up to 39.42% compared with state-of-the-art utility-agnostic methods. Moreover, 04A is robust to estimation errors in job processing time and transmission delay.

## CCS CONCEPTS

• **Networks** → **Network algorithms**; **Network protocols**.

## KEYWORDS

Online Approximation, Job Scheduling, Heterogeneous Utility Function, Edge Computing

This work is supported partly by the National Key R&D Program of China 2018YFB0803400, NSFC Grants 61772489, 61751211, Key Research Program of Frontier Sciences (CAS) No. QYZDY-SSW-JSC002, the Anhui Dept. of Sci. and Tech. under grant 201903a05020049NSF, CNS 1526638, and Microsoft Research Asia. Haisheng Tan is the corresponding author (Email: hstan@ustc.edu.cn).


*Mobihoc '20, October 11–14, 2020, Boston, MA, USA*

ACM ISBN 978-1-4503-8015-7/20/10…$15.00
https://doi.org/10.1145/3397166.3409122

**ACM Reference Format:**
Chi Zhang, Haisheng Tan, Haoqiang Huang, Zhenhua Han, Shaofeng H.-C. Jiang, Nikolaos Freris, and Xiang-Yang Li. 2020. Online Dispatching and Scheduling of Jobs with Heterogeneous Utilities in Edge Computing. In *The Twenty-first ACM International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (Mobihoc '20), October 11–14, 2020, Boston, MA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397166.3409122

## 1 INTRODUCTION

Emerging applications in the era of 5G, such as virtual/augmented reality and autonomous driving, require low-latency access to powerful computation resources [9, 23]. Due to long distances from remote cloud data-centers, along with limited wide area network bandwidths, edge computing is rising as a promising technique 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, thus enlarging the coverage of cloud computing.

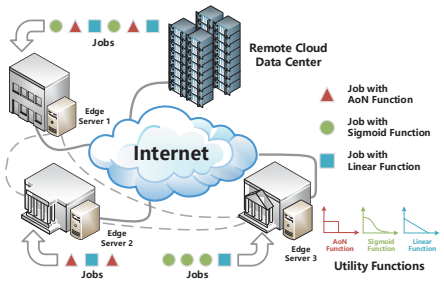



**Figure 1: Illustration of an edge-cloud system with mobile devices, edge servers, the remote cloud, alongside jobs with heterogeneous utility functions.**

In order to efficiently utilize the limited resources in edge-clouds, one fundamental problem 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 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 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 [24, 30, 31, 35] were mainly dedicated to optimizing a single utility function among all jobs. Nevertheless, varying the utility function different scheduling disciplines, whence utility-agnostic schedulers might fail to take advantage of the computational resources in edge systems to maximize the aggregate utility over all jobs.

In this paper, we study online dispatching and scheduling of jobs with heterogeneous utility functions in edge-cloud systems. We consider a general setting, in which there is a set of online jobs that may arrive at arbitrary times and in arbitrary order. Each job is associated with a utility function. A variety of jobs coexist in the system which are allowed to have different utility functions. Here, a utility function can be in any form as long as non-increasing with respect to the JRT. We adopt the unrelated machine model, where a job may have variable and server-independent execution time, uploading time, and downloading time across different servers. Our goal is to maximize the total utility obtained for all jobs.

We use the *competitive ratio*, the largest possible ratio between the algorithm performance and the offline optimal, as a metric to evaluate the theoretical performance of our method. 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 [10]. Thus, we adopt the speed augmentation model [4] by allowing the edge servers to be $(1+\epsilon)$ times as fast as those in the optimal offline solution, where $\epsilon$ is a small positive constant [1]. Our main contributions are summarized as:

- We formulate a novel online job dispatching and scheduling problem in edge computing under a general model, where different jobs may employ heterogeneous utility functions. Additionally, the upload/download delay and the unrelated-machine job processing time are considered (Sec. 3).
- We further propose an online algorithm, named 04A (standing for "One algorithm for All utility functions"), and prove that it has a competitive ratio of $O(\frac{1}{\epsilon^2})$ (Sec. 4).
- We implement 04A on a small-scale testbed consisting of 20 edge devices. Our experiments show that 04A can increase the aggregate job utility by up to 39.42% compared with state-of-the-art baselines. Besides, the gap between 04A and the optimal solution is within 10.40% (Sec. 5).
- Through extensive large-scale simulations on the production trace from Google, we demonstrate that 04A consistently outperforms the baselines over various workloads and parameters. Moreover, 04A is robust to estimation errors in job processing time and the communication delay (Sec. 5).

[1] In practice, the speed augmentation analysis can be understood as follows: our algorithm achieves the performance approximation ratio, as long as the capability of servers is upgraded by a small factor $\epsilon$.

## 2 MOTIVATION

We first show the prevalence of heterogeneous utilities in edge applications. Then, we explain why existing schedulers could perform poorly if they are unaware of the heterogeneity.

### 2.1 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 multiple 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 imposes strict requirements on the response latency, *e.g.*, 100 ms as mentioned in [22]. Its utility function can be represented with an All-or-Nothing (AoN) function (as shown in Figure 1).
- **Driver Assistance**: To improve the driving safety, machine learning techniques are used for assisting drivers to avoid potential risks, which include fine-grained face recognition [34], body pose estimation [27], semantic scene perception, and driving state prediction [37]. The utility function could be sigmoid-like, whose QoE is sensitive to a range of latency [38].
- **Data Preprocessing**: Autonomous vehicles are estimated to generate as much as 4 terabytes of data per hour [36]. 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.

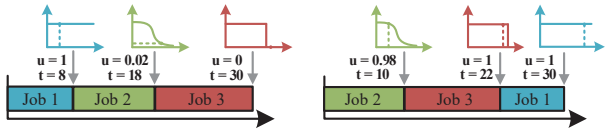
In addition to autonomous driving, public edge-clouds might need to support a wide range of applications, *e.g.*, content delivering, video streaming, and IoT analytics, which can employ various sensitivity to latency. To efficiently utilize limited edge resources, edge schedulers should allow edge applications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

### 2.2 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 [31] and Dedas [24]. We take the shortest job first (SJF) policy adopted by OnDisc [31] 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 arrives at time 0. Their processing times are $p_1 = 8, p_2 = 10$ and $p_3 = 12$, respectively. Their utility functions are $g_1(t) = 1, g_2(t) = \frac{1}{1+e^{t-14}}$ and $g_3(t) =$

$\begin{cases} 1 & t \le 25 \\ 0 & t > 25 \end{cases}$, which are the constant, sigmoid 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 $1 + 0.02 + 0 = 1.02$. The optimal solution (shown in Fig. 2 (b)) is in the order of $j_2, j_3, j_1$ with a total utility of $0.98 + 1 + 1 = 2.98$. 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 algorithms and system implementation.




(a) SJF Scheduler (b) Optimal

**Figure 2: Inefficiency of the SJF when heterogeneous utility functions coexist.**

## 3 SYSTEM MODEL

**Edge System:** We consider an edge-cloud system with a number of heterogeneous edge servers and a remote cloud, denoted as set $\mathcal{S}$, where the remote cloud is also regarded as a special server with long transmission latency and 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 times, and one may assume no prior information before the job release. To avoid migration overhead, a server can not migrate jobs to the others after the job dispatching decision. Preemption is allowed so that an executing job might be halted and the server can resume its processing later.

**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 $i$ can not 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. 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.

**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 its sensitivity to the JRT. Generally, a utility function can be any non-increasing function, which means we can not gain a higher utility by postponing the job's processing.

**Problem Formulation:** With the aforementioned model, our problem is to design online dispatching and scheduling for jobs with heterogeneous utilities aiming at maximizing the total utility, *i.e.*, $\sum_{i\in\mathcal{J}} g_i(c_i)$. The problem is formulated as:

Problem 1.

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

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

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

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

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

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

*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, which maximizes the total utility of all jobs. Constraint (4) guarantees each server only processes one job at any time. Constraint (5) guarantees each job is dispatched to one server. Constraint (6) guarantees each job is processed for at least $p_{i,j}$.*

**Hardness:** The problem 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 [2]. We here employ the speed augmentation model in our analysis. Formally, we define $(1+\epsilon)$-speed augmentation [4]:

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

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

Definition 2. *An online algorithm is $(1+\epsilon)$-speed $c$-competitive if for any job sets, we have $\frac{OPT}{ALG} \le c$, where* ALG *denotes the utility gained by the online algorithm with $(1+\epsilon)$-speed and* OPT *denotes the total utility of the optimal solution on the original servers.*

Furthermore, we derive the following lower bound on the competitive ratio in the speed augmentation model for Problem 1. Due to the limited space, the detailed proof is omitted.

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

## 4 THE 04A ALGORITHM

In this section, we present our online algorithm 04A, to solve the joint dispatching and scheduling problem in edge systems featuring

jobs with heterogeneous utility functions, and analyze its competitive ratio theoretically.

## 4.1 General Idea

For each job dispatched to a server, 04A 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, 04A adopts a parameter to reserve more execution time. Given a job's utility function, 04A calculates the potential time intervals on its dispatched server that can be used to execute this job. At any time, 04A 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, 04A 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 04A in details.

## 4.2 Tentative Execution

04A leverages the speed augmentation parameter $\epsilon$ when making scheduling decisions. Assuming the job $i$ is dispatched to server $j$, we define the job's *tentative execution time* as $p^*_{i,j} = \frac{1+2\epsilon}{1+\epsilon} p_{i,j}$, which is a conservative processing time that 04A 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 $\frac{2\epsilon}{1+\epsilon} p_{i,j}$ in $p^*_{i,j}$ is the time reserved by 04A for addressing potential preemption by future-arriving jobs. When planing the job's execution, 04A tentatively takes $p^*_{i,j}$ as the job's processing time, and thus determines its *tentative completion time* (denoted by $d^*_{i,j}$), *i.e.*, the time when the job $i$ is processed for a duration of $p^*_{i,j}$.

## 4.3 Computing a Tentative Execution Plan

Following the definition of tentative execution, 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 densities is approximately equal to the number of queueing 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 3–4). As we will prove later, this optimization does not hurt the asymptotic performance of 04A, 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, 04A schedules jobs following a highest utility density-first principle over all unfinished jobs. Since a job's utility density depends on its tentative completion time $d^*_{i,j}$, Algorithm 1 computes the minimum tentative completion time $d^*_{i,j}$ 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 densities on server $j$ in ascending order, so as to compute the priority assigned to job $i$. For the $\gamma$-th utility density $u_\gamma$, we calculate job $i$'s tentative completion time if job $i$ is assigned with the $\gamma$-th priority (Line 13–16). The

**Algorithm 1:** Tentative Schedule Planning

1 **Input** *job i, server j*;
2 $\alpha = 1 + \frac{1}{\epsilon}$ ;
3 $\lambda^*_i(t) = \max\{\lambda | p_{i,j} \cdot 2^\lambda \le g_i(t), \lambda \text{ is an integer}\}$;
4 $g^D_i(t) = p_{i,j} \cdot 2^{\lambda^*_i(t)}$;
5 $\mathcal{J}_Q$ = the set of unfinished jobs at server $j$;
6 Define $u_{i',j} = \frac{g^D_{i'}(d^*_{i',j} + \Delta^{\downarrow}_{i',j} - r_i)}{p_{i',j}}$ as the utility density of job $i' \in \mathcal{J}_Q$;
7 Let the distinct utility densities in $\mathcal{J}_Q$ be $u_1, u_2, \ldots, u_K$ (in ascending order);
8 The set of feasible tentative densities $\mathcal{U} = \varnothing$;
9 **for** $\gamma = 0$ *to* $K$ **do**
10  **if** $\gamma = 0$ **then**
11  $\mathcal{I}^\gamma_{i,j} = \{t : t \ge r_i + \Delta^{\uparrow}_{i,j}, t \notin \cup_{i' \in \mathcal{J}_Q} \mathcal{I}_{i'}\}$;
12  **else**
13  $i^\gamma$ = the $\gamma$-th job in $\mathcal{J}_Q$;
14  $\mathcal{J}^\gamma_Q = \{i' : i' \in \mathcal{J}_Q, u_{i',j} > u_\gamma\}$;
15  $\mathcal{I}^\gamma_{i,j} = \{t : t \ge r_i + \Delta^{\uparrow}_{i,j}, t \notin \cup_{i' \in \mathcal{J}^\gamma_Q} \mathcal{I}_{i'}\}$;
16  $d^\gamma_{i,j} = \min\{t | \int_{r_i}^{t} \mathbf{1}[t \in \mathcal{I}^\gamma_{i,j}] dt \ge p^*_{i,j}\}$;
17  $u^\gamma_{i,j} = \frac{g^D_i(d^\gamma_{i,j} + \Delta^{\downarrow}_{i,j} - r_i)}{p_{i,j}}$;
18  **if** $u^\gamma_{i,j} > \alpha u_{i',j}$ *for all* $i' \in \mathcal{J}_Q - \mathcal{J}^\gamma_Q$ **then**
19  Add $u^\gamma_{i,j}$ to $\mathcal{U}$;
20 $\gamma^* = \arg\max_\gamma \mathcal{U}$;
21 $\mathcal{I}_{i,j} = \mathcal{I}^{\gamma^*}_{i,j}$; $d^*_{i,j} = d^{\gamma^*}_{i,j}$; $u_{i,j} = u^{\gamma^*}_{i,j}$;

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 $u_\gamma$, *i.e.*, $\mathcal{J}^\gamma_Q$. To achieve this, we use *segment tree optimization*, maintaining a segment tree for each $\gamma$ to store all intervals with utility density greater than $u_\gamma$. 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 $\alpha$ as a preemption threshold: job $i$ is placed at the $\gamma$-th position when its utility density is higher than $\alpha$ times that of any other job with lower priorities. In other words, 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 shows an example of how Algorithm 1 operates. At the beginning, there are three unfinished jobs on the server: Job 1, 2 and 3 in ascending order in terms of the utility density. Job 2 has a higher density than Job 1, but less than $\alpha$ times, so Job 2's executable time cannot overlap with Job 1's executable time. For the same reason, Job 3's executable time cannot overlap with Job 2's. Because Job 3's utility density is higher than $\alpha$ times of Job 1's density, Job 3' executable time can overlap with Job 1's so that Job 3 can preempt Job 1 when it arrives on the server. When the newly released Job

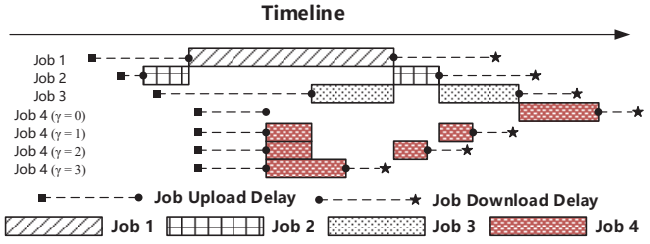



**Figure 3: An example of how Algorithm 1 comoutes the tentative schedule for a newly released job (Job 4).**

**Algorithm 2:** Dispatching and Scheduling Policy in 04A

1 **Job Dispatching:** When job $i$ is released at time $r_i$, it is dispatched to server $j = \arg\min_{j' \in \mathcal{S}}\{d^*_{i,j'} + \Delta^{\downarrow}_{i,j'}\}$ ;

2 Determine the tentative finish time of job $i$: $f_i = d^*_{i,j} + \Delta^{\downarrow}_{i,j}$ ;

3 **Job Scheduling:** At time $t$, server $j$ schedules the executable job $i$ with the highest utility density,*i.e.*, $i = \arg\max_{\{i' | t \in \mathcal{I}_{i',j}\}} u_{i',j}$ ;

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 other three jobs finish 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. 04A will choose the one that maximizes its utility.

## 4.4 Job Scheduling & Dispatching

Based on 04A's tentative schedule planning, we elaborate in Algorithm 2 how 04A 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, 04A 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 afterwards.

**Time Complexity:** Let $K$ be the number of distinct utility densities 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 04A is $O(K \cdot L \cdot \log^2 T)$.

## 4.5 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 04A 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 04A 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 using amortized analysis.
- In the Lemma 2 and Lemma 3, we upper-bound $OPT^D$ by the cumulative tentative utility.
- 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 $f_i \triangleq d^*_{i,j} + \Delta^{\uparrow}_{i,j}, \mathcal{I}_i \triangleq \mathcal{I}_{i,j}$ be the tentative finish time and tentative interval of job $i$ if job $i$ is dispatched to server $j$, respectively. We let $w^D_i \triangleq g^D_i(f_i - r_i)$ be the discretized tentative utility, and let $u^D_i \triangleq \frac{w^D_i}{p_{i,j}}$ be the discretized tentative density. Finally, we denote the set of all jobs by $\mathcal{J}$, and use $C$ for the set of jobs that finish prior to their tentative finish time as calculated by 04A.

Lemma 1. *The utility obtained from* 04A *is at least* $(1 - \frac{1+2\epsilon}{2\epsilon(\alpha-1)})$ *times the total utility if each job $i$ was completed at time $f_i$,* i.e., $ALG \geq (1 - \frac{1+2\epsilon}{2\epsilon(\alpha-1)}) \sum_{i \in \mathcal{J}} w^D_i$.

Proof. For each server $j$, we will use the charging scheme transfer utility among jobs. Let non-negative $h^{in}_i$ be the utility transfer to job $i$ and $h^{out}_i$ be the utility transfer out from job $i$. For any time $t$ and server $j$, let $X_{t,j}$ be the set of jobs whose interval $\mathcal{I}_i$ contains $t$. The job $i$ with the highest density $u^D_i = \frac{w^D_i}{p_{i,j}}$ in $X_{t,j}$ transfers utility to other jobs in $X_{t,j}$. For each job $i' \in X_{t,j} - \{i\}$, the transfer speed is $(\frac{1+\epsilon}{2\epsilon})\frac{w^D_{i'}}{p_{i',j}}$. Note that the total utility of all jobs will not change, that is $\sum_{i \in \mathcal{J}} h^{in}_i = \sum_{i \in \mathcal{J}} h^{out}_i$. Thus, we have $ALG = ALG + \sum_{i \in \mathcal{J}} h^{in}_i - \sum_{i \in \mathcal{J}} h^{out}_i$. By the definition of $C$ and $w^D_i$, we have $ALG \geq \sum_{i \in C} w^D_i$. For each job $i$ which is not completed, there must be some more dense jobs processing at least $\frac{2\epsilon}{1+\epsilon} p_{i,j}$ units of time in interval $\mathcal{I}_i$. That is $h^{in}_i \geq \frac{1+\epsilon}{2\epsilon}\frac{w^D_i}{p_{i,j}} \cdot \frac{2\epsilon}{1+\epsilon} p_{i,j} = w^D_i$, $i \in \mathcal{J} - C$. Thus, $\sum_{i \in \mathcal{J}-C} h^{in}_i \geq \sum_{i \in \mathcal{J}-C} w^D_i$.

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 $u^D_i > \alpha u^D_{i'}$ or $u^D_{i'} > \alpha u^D_i$. Thus at any time $t$ the speed of utility transferred from job $i$ is at most $(\frac{1+\epsilon}{2\epsilon})\frac{w^D_i}{p_{i,j}} \sum_{k=1}^{\infty} \frac{1}{\alpha^k} \leq (\frac{1+\epsilon}{2\epsilon(\alpha-1)})\frac{w^D_i}{p_{i,j}}$. The total length of intervals in $\mathcal{I}_i$ is $\frac{1+2\epsilon}{1+\epsilon} p_{i,j}$, thus $h^{out}_i \leq (\frac{1+\epsilon}{2\epsilon(\alpha-1)})\frac{w^D_i}{p_{i,j}} \cdot \frac{1+2\epsilon}{1+\epsilon} p_{i,j} = \frac{1+2\epsilon}{2\epsilon(\alpha-1)} w^D_i$. Thus $\sum_{i \in \mathcal{J}} h^{out}_i \leq \sum_{i \in \mathcal{J}} \frac{1+2\epsilon}{2\epsilon(\alpha-1)} w^D_i$. Finally, we have

$$\begin{aligned} ALG &\geq \sum_{i \in C} w^D_i + \sum_{i \in \mathcal{J}-C} w^D_i - \frac{1+2\epsilon}{2\epsilon(\alpha-1)} \sum_{i \in \mathcal{J}} w^D_i \\ &\geq \left(1 - \frac{1+2\epsilon}{2\epsilon(\alpha-1)}\right) \sum_{i \in \mathcal{J}} w^D_i \end{aligned}$$

, which concludes our proof. □

Let $A^*_1$ be the set of jobs such that $i \in A^*_1$ implies that the tentative time $t_i$ set by the algorithm is no later than $t^*_i$, the completion 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 subset of $A^*_1, A^*_2$, respectively. 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 $A^*_2$ at optimality. 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 are 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 process a job with density at least $\frac{u}{\alpha}$ on server $j$.

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

Proof. Fix a server $j$. For each job $i$, we define a window $[r_i + \Delta^{\uparrow}_{i,j}, f^*_i - \Delta^{\downarrow}_{i,j}]$, where $r_i$ is the release time of job $i$ and $f^*_i$ is the finish time in optimal scheduling. Let $W^*_j$ be the set of all the windows of jobs with density at least $u$ in $A^*_2$. For ease of analysis, let $M_j$ be the minimal subset of $W^*_j$ whose windows span every time contained in a window in $W^*_j$. Then we can divide all the windows of $M_j$ into two sets, *i.e.*$M^1_j$ and $M^2_j$, each of which does not have overlapping windows. Without lose of generality, we assume that the total length of time contained in a window of $M^1_j$ is at least that of $M^2_j$. Then we have the total length of time contained in a window of $M^1_j$ is at least half of that of $W^*_j$. 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 $\frac{u^D_i}{\alpha}$ in interval $[r_i + \Delta^{\uparrow}_{i,j}, f^*_i - \Delta^{\downarrow}_{i,j}]$ is at least $\frac{2\epsilon}{1+\epsilon}[f^*_i - \Delta^{\downarrow}_{i,j} - (r_i + \Delta^{\uparrow}_{i,j})]$. Furthermore, $L_j(\frac{u}{\alpha})$ is at least $\frac{2\epsilon}{1+\epsilon}\times$ the total length of time contained in a window of $M^1_j$, *i.e.*, $\frac{\epsilon}{1+\epsilon}\times$ the total length of time contained in a window of $W^*_j$. The proof is concluding by nothing that, by definition, $L^*_j(u)$ is not larger than the total length of time contained in a window in $W^*_j$. □

Lemma 3. $\sum_{i\in A^*_2} w_i^{D*} \leq \frac{\alpha(1+\epsilon)}{\epsilon} \sum_{i\in\mathcal{J}} w_i^D$.

Proof. The proof is similar to Lemma 3.5 in [17], we skip the details here. □

Theorem 2. *For any $\epsilon \in (0, \frac{1}{2})$, 04A is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive.*

Proof. By using lemma 1 and 3, we have

$$OPT^D = \sum_{i\in A^*_1\cup A^*_2} w_i^{D*} \leq \sum_{i\in A_1} w_i^D + \frac{\alpha(1+\epsilon)}{\epsilon} \sum_{i\in\mathcal{J}} w_i^D$$

$$\leq (1 + \frac{\alpha(1+\epsilon)}{\epsilon}) \sum_{i\in\mathcal{J}} w_i^D \leq \frac{1 + \frac{\alpha(1+\epsilon)}{\epsilon}}{1 - \frac{1+2\epsilon}{2\epsilon(\alpha-1)}} ALG.$$

Using $\alpha = 1 + \frac{1}{\epsilon}$, we deduce $OPT^D \leq \frac{2+4\epsilon+4\epsilon^2}{\epsilon^2-2\epsilon^3} ALG$. Since all the utilities decrease by at most $\frac{1}{2}$ of the original after discretization, we have $OPT^D \geq \frac{1}{2} OPT$. Thus $ALG \geq \frac{\epsilon^2-2\epsilon^3}{4+8\epsilon+8\epsilon^2} OPT$. □

## 5 PERFORMANCE EVALUATION

In this section, we evaluate 04A's performance using production-trace driven testbed experiments and large-scale simulations. Overall, our key findings are:

**Table 1: Inference time of five popular ML models**

| Model | Nvidia Jetson Nano (MXNet v1.4.1) | Raspberry Pi 3b (MXNet v1.5.0) |
|---|---|---|
| DenseNet-121 [15] | 223ms | 247ms |
| DenseNet-169 | 277ms | 342ms |
| MobileNet 0.25 [14] | 108ms | 62ms |
| MobileNet 0.5 | 109ms | 62ms |
| MobileNetV2 0.25 [28] | 137ms | 103ms |
| MobileNetV2 0.5 | 137ms | 104ms |
| ResNet-18 V1 [12] | 105ms | 53ms |
| ResNet-18 V2 [13] | 100ms | 51ms |
| SqueezeNet 1.0 [16] | 95ms | 54ms |

- In the testbed experiments running deep learning inference jobs, 04A performs 13.49% and 39.42% better than OnDisc and Random, respectively, when there are multiple types of utility functions. Besides, the total utility obtained by 04A is close to the optimal with a gap no larger than 10.40%.
- When the job transmission time and processing time are misestimated, 04A is more robust compared with the baseline algorithms. In specific, the average utility is degraded by at most 6.26% under 20% estimation error.
- Extensive simulations show that 04A is robust to various workload and parameter variations and consistently performs better than state-of-the-art baselines.

### 5.1 Experimental Settings

**Testbed Cluster:** Our edge-cloud testbed consists of 20 workers: 10 Nvidia Jetson Nano's and 10 Raspberry Pi 3b'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 3b has 4 cores, 1 GB RAM, 16 GB storage and a 100 Mbps 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.

**Workloads:** We choose deep learning inference jobs as our workloads to evaluate 04A and the baseline algorithms. Table 1 lists the deep learning models of the jobs and their performance on Nvidia Jetson Nano and Raspberry Pi 3b. Specifically, each job requests to classify an image from ImageNet [8] using one of the models listed in Table 1. The release times of jobs are set based on the trace of Google's production clusters [26]. In the testbed experiments, the average job release rate is ~ 15 jobs per second. Note that since Nvidia Jetson Nano and Raspberry Pi 3b have different hardware specifications and running 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.

**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 [5], 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).

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

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.

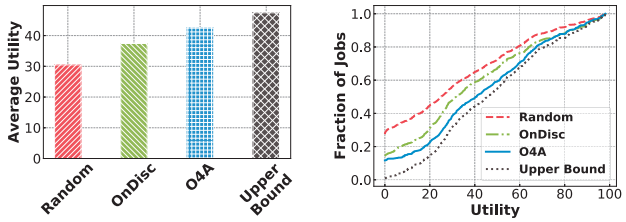
**Baselines:** We compare O4A with the following baselines:

- Random: dispatching jobs to servers randomly. On each server, the job with the highest potential utility is picked to process.
- OnDisc [31]: dispatching a job to the server which causes the least increase in job completion time. On each server, the scheduling discipline of shortest remaining processing time-first is followed.
- Upper Bound: For any job $i$, its possible utility can not 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.

## 5.2 Testbed Experiment: Overall Comparison

**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 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 13.49% and 39.42% higher utility than OnDisc and Random, respectively. The CDF of utility (shown in 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 10.40%, which reveals that the total utility obtained by O4A is close to the optimal utility.

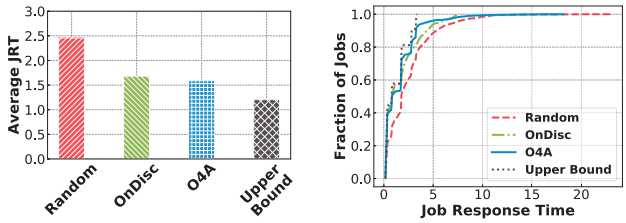
**Improvement on Job Response Time:** Fig. 5 presents the overall average JRT and CDF of JRT in the testbed experiment. Although OnDisc is designed for optimizing job completion time, the average JRT of O4A's is still 5.51% lower than OnDisc's, cf. Fig. 5(a). OnDisc's degraded performance is due to the variance of job processing time and upload/download delay. Additionally, inappropriate dispatching caused by misestimation may result in unbalanced server load and thus the server resources can not be fully utilized. However, 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




(a) Average Utility (b) Distribution of Utility

**Figure 4: Average Utility and Distribution of the Utility (All Utility Functions).**

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 algorithm.

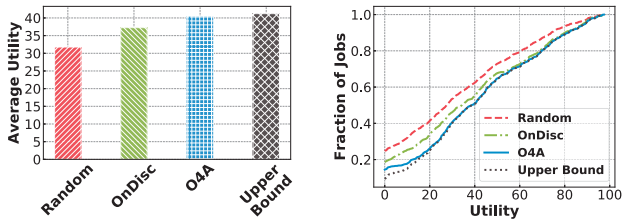



(a) Average JRT (b) Distribution of JRT

**Figure 5: Average JRT and Distribution of JRT (All Utility Functions).**

## 5.3 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 for all algorithms.

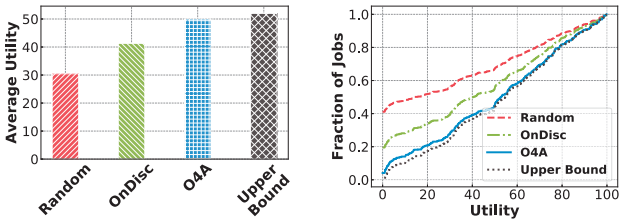



(a) Average Utility (b) Distribution of Utility

**Figure 6: Average Utility and Distribution of Utility (Linear Utility Functions Only).**

**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. O4A still performs better than the other two baselines on all percentiles. OnDisc features

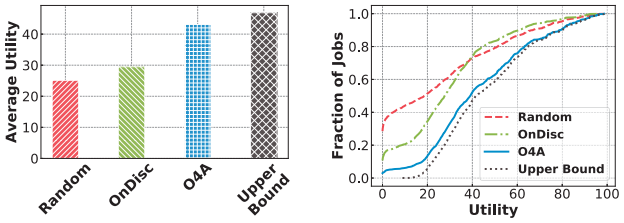
similar performance with O4A on the percentiles over 70%, while it performs worse on the lower percentiles. This can be understood since OnDisc's greedy scheduling favors high-utility jobs too much, whence it typically fails to achieve optimal trade-off between the high utility jobs and the low utility jobs.




(a) Average Utility (b) Distribution of Utility

**Figure 7: Average Utility and Distribution of Utility (AON Utility Functions Only).**

**Impact of AoN Function:** Fig. 7 shows average utility and utility CDF for all algorithms when only submitting jobs with AoN utilities. AoN functions allow the jobs to express their deadlines. O4A not only achieves higher utility, but also 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 only misses 4.11% job deadlines, while OnDisc and Random miss 19.51% and 41.12% jobs, respectively. Since both baseline algorithms are deadline-agnostic (they only seek to optimize job completion time), they may waste workers' time in processing jobs whose deadline can hardly be satisfied.




(a) Average Utility (b) Distribution of Utility

**Figure 8: Average Utility and Distribution of Utility (Sigmoid Utility Functions Only).**

**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 functions, compared with linear and AoN functions: O4A achieves 71.58% and 46.16% higher utility than Random and OnDisc, respectively. The degraded performance of the two baselines is mainly due to the non-linearity of the sigmoid function.

## 5.4 Large-scale Simulations

We conduct sensitivity studies by large-scale simulations in a cluster of 500 edge servers based on production traces. Specifically, we investigate the impact of the job release rate, the number of edge servers, the misestimation on job information, and the speed augmentation parameter $\epsilon$.

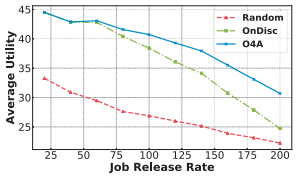



**Figure 9: Impact of the Job Release Rate.**

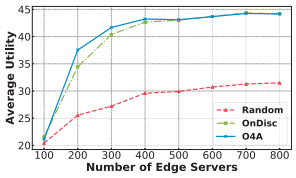



**Figure 10: Impact of the Number of Servers.**

**Impact of Job Release Rate:** We present the trend with rate changes in Fig. 9, 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.

**Impact of the Number of Edge Servers:** Fig. 10 shows the trend of the average utility with variable number of edge servers. All algorithms benefit from adding edge servers. Random only obtains a marginal increase on the average utility, while both OnDisc and O4A attain higher gains.

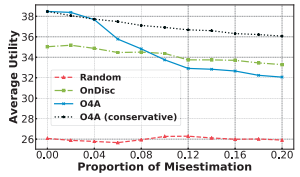



**Figure 11: Impact of the Misestimation.**

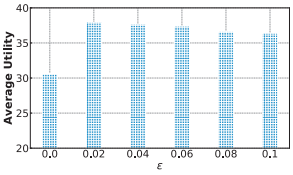



**Figure 12: Impact of Parameter $\epsilon$.**

**Impact of Inaccurate Estimation:** In an actual production environment, it is hard to acquire accurate estimation of the job processing time or the transmission delay. Fig. 11 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 $\delta$. 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 $\delta$ 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 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 more robust to estimation errors, even when the misestimation is up to 20%.

**Impact of the speed augmentation parameter $\epsilon$:** In Fig. 12, we study the impact of varying the speed augmentation parameter $\epsilon$ from 0 to 0.1. We unravel that altering the value of $\epsilon$ has little impact on O4A's performance. Recall that O4A's parameter $\alpha = 1+\frac{1}{\epsilon}$, 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 04A from becoming a non-preemptive scheduler.

## 6 DISCUSSION

**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 04A uses a preemption knob $\alpha$ 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 $\alpha$, 04A can adjust its switching frequency taking into consideration the switching overhead.

**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. 5.4, we evaluate the impact of estimation error and show that a simple modification to 04A greatly improves its robustness. It is interesting to theoretically study the impact of estimation error to 04A's competitive ratio, and this will be a focal point for future research.

**Comparison with Utility-Specific Algorithms:** 04A is designed for general heterogeneous utility functions (*i.e.*, each job may choose a different type of utility function, with different parameters), which may sacrifice performance for generality. In the worst case, the competitive ratio of 04A is $O(\frac{1}{\epsilon^2})$, which can be worse than algorithms designed for a specific type of utility function. For instance, OnDisc [31] is $O(\frac{1}{\epsilon})$-competitive, for the linear utility only. However, our evaluation based on production traces reveals that 04A can achieve comparable performance to utility-specific algorithms. In conclusion, we attest that 04A reaches a good trade-off between performance and generality in utility selection.

**Distributed Implementation:** The original version of 04A appears to be centralized. Notwithstanding, capitalizing on the attributes of 04A, we may devise an implementation in a distributed environment via the following three steps:

(1) When job $i$ is released at time $r_i$, it sends request messages with its job information to each possible server $s \in \mathcal{S}$, 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.
(3) Job $i$ picks the server with the earliest tentative finish time for dispatching.

Since the remote cloud suffers from long communication latencies, we avoid obtaining the tentative finish time on the remote cloud server. Instead, as the remote cloud has abundant resources compared with the edge, we can infer its tentative finish time locally by assuming there is no contention delay.

## 7 RELATED WORK

**Cloud and Edge Computing:** Since cloud computing is one of the most popular computing paradigms, there are plentiful works (*e.g.*, [7, 11, 21]) focusing on offloading jobs from the mobile devices to the cloud data center. However, communication with the remote cloud might incur inevitable long communication delay. By deploying edge servers near mobile users, edge computing can provide users with rich computation resources with low communication latency [29]. Edge computing injects vitality into the filed of mobile computing, but it also brings challenges.

Since resources in edge servers are relatively limited, many studies focusing on resource management and load balancing in edge computing. With the practical assumption that user locations might keep changing, Urgaonkar *et al.* [33] modeled the workload scheduling problem as Markov Decision Process and adopted Lyapunov optimization to solve the problem. Tong *et al.* [32] designed a tree hierarchical edge-cloud and proposed an efficient heuristic workload dispatching policy to offload the workload. More recently, Meng *et al.* [24] took the network bandwidth into consideration and proposed a deadline-aware job dispatching strategy. All the aforementioned work relies on the stochastic optimization and assume that the job releasing pattern follows some specific distribution, which might not hold in practice. Instead, Tan *et al.* [31] investigated *online* job dispatching and scheduling in edge computing, where jobs arrive at arbitrary time and order. They proposed an efficient online algorithm to minimize the total weighted job response time, which achieves a competitive ratio of $O(\frac{1}{\epsilon})$ under the $(1+\epsilon)$-speed augmentation model. As stated above, previous works mainly focused on a single specific metric. However, in practice, a variety of applications can coexist, 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 are allowed to co-exist in the system.

**Classic Online Scheduling Problem:** In online theory, scheduling a set of jobs for utility maximization has been studied for decades. One typical setting is that each job arrives online with a utility and a deadline. If the job can complete before its deadline, we earn its utility; otherwise, the utility is zero. Here, the job density is defined as the utility divided by its processing time. In the case of a single server and preemption allowed, Sanjoy *et al.* [2, 3, 20] conducted a series of influential works, where they resolved the problem complexity and figured out the optimal deterministic online algorithm. However, the competitive ratio is $\Theta(K)$, where $K$ is the ratio of the maximum to the minimum job density, which depends on the instance of the set of jobs and can be extremely large. Gilad *et al.* [19] and Bala *et al.* [18] employed the power of randomization and derived the optimal randomized competitive ratio as $\Theta(\min\{\log K, \log \psi\})$ where $\psi$ is the ratio of the maximum to the minimum job size, which is still related to the input instance. Due to the hardness of approximation, some works [1, 6, 17, 25] resorted 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.* [25] constructed a $(1+\epsilon)$-speed $O(1)$ competitive algorithm for any fixed constant $\epsilon > 0$ in the identical severs setting. 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. In our algorithm 04A, 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, 04A is the first practical algorithm to dispatch and schedule jobs with heterogeneous utilities in edge computing.

## 8 CONCLUSION

Motivated by piratical applications such as autonomous driving, we studied online dispatching and scheduling of jobs with heterogeneous utility functions in edge computing. We here adopt a general model where each coexisting job in an edge system could employ any non-increasing utility function over its job response time. Moreover, the unrelated machine model and upload and download delay are considered. We devised 04A, an online algorithm to maximize the total utility obtained from all jobs, and theoretically established that it is $O(\frac{1}{\epsilon^2})$-competitive under the $(1+\epsilon)$-speed augmentation model. Here, $\epsilon$ is a small positive constant. Moreover, we implemented 04A on a small-scale edge computing testbed running widely used machine learning inference tasks. Our experiments and extensive production trace-driven simulations illustrate that 04A can increase the total utility by up to 39.42% compared with state-of-the-art baselines, while also achieving comparable or better average job response time and deadline miss ratio. 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 of the current work is to theoretically analyze the impact of estimation error on the performance of the online dispatching and scheduling strategies.

## REFERENCES


[1] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. 2011. Competitive algorithms for due date scheduling. *Algorithmica* 59, 4 (2011), 569–582.

[2] Sanjoy Baruah, Gilad Koren, Decao Mao, Bhubaneswar Mishra, Arvind Raghunathan, Louis Rosier, Dennis Shasha, and Fuxing Wang. 1992. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems* 4, 2 (1992), 125–144.

[3] Sanjoy Baruah, Gilad Koren, Bhubaneswar Mishra, Arvind Raghunathan, Louis Rosier, and Dennis Shasha. 1991. On-line scheduling in the presence of overload. In *Proc. of FOCS*.

[4] Jivitej S Chadha, Naveen Garg, Amit Kumar, and VN Muralidhara. 2009. A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation. In *Proc. of STOC*.

[5] Li Chen, Wei Cui, Baochun Li, and Bo Li. 2016. Optimizing coflow completion times with utility max-min fairness. In *Proc. of INFOCOM*.

[6] Lin Chen, Franziska Eberle, Nicole Megow, Kevin Schewior, and Cliff Stein. 2019. A general framework for handling commitment in online throughput maximization. In *Proc. of IPCO*.

[7] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. 2011. Clonecloud: elastic execution between mobile device and cloud. In *Proc. of EuroSys*.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *Proc. of CVPR*.

[9] Jingyun Feng, Zhi Liu, Celimuge Wu, and Yusheng Ji. 2017. AVE: Autonomous vehicular edge computing framework with ACO-based scheduling. *IEEE Transactions on Vehicular Technology* 66, 12 (2017), 10660–10675.

[10] Naveen Garg and Amit Kumar. 2007. Minimizing average flow-time: Upper and lower bounds. In *Proc. of FOCS*.

[11] Mark S Gordon, D Anoushe Jamshidi, Scott Mahlke, Z Morley Mao, and Xu Chen. 2012. COMET: Code Offload by Migrating Execution Transparently. In *OSDI*.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proc. of CVPR*.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *Proc. of ECCV*.

[14] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[15] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proc. of CVPR*.

[16] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[17] Sungjin Im and Benjamin Moseley. 2016. General profit scheduling and the power of migration on heterogeneous machines. In *Proc. of SPAA*.

[18] Bala Kalyanasundaram and Kirk Pruhs. 2000. Fault-tolerant real-time scheduling. *Algorithmica* 28, 1 (2000), 125–144.

[19] Gilad Koren and Dennis Shasha. 1994. MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. *Theoretical Computer Science* 128, 1-2 (1994), 75–97.

[20] Gilad Koren and Dennis Shasha. 1995. D^over: An Optimal On-Line Scheduling Algorithm for Overloaded Uniprocessor Real-Time Systems. *SIAM J. Comput.* 24, 2 (1995), 318–339.

[21] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. 2012. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proc. of INFOCOM*.

[22] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. 2018. The architectural implications of autonomous driving: Constraints and acceleration. In *ACM SIGPLAN Notices*, Vol. 53. ACM, 751–766.

[23] Luyang Liu, Ruiguang Zhong, Wuyang Zhang, Yunxin Liu, Jiansong Zhang, Lintao Zhang, and Marco Gruteser. 2018. Cutting the cord: Designing a high-quality untethered vr system with low latency remote rendering. In *Proc. of Mobicom*.

[24] Jiaying Meng, Haisheng Tan, Chao Xu, Wanli Cao, Liuyan Liu, and Bojie Li. 2019. Dedas: Online Task Dispatching and Scheduling with Bandwidth Constraint in Edge Computing. In *Proc. of INFOCOM*.

[25] Kirk Pruhs and Cliff Stein. 2010. How to schedule when you have to buy your energy. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 352–365.

[26] Charles Reiss et al. 2011. *Google cluster-usage traces.* Technical Report.

[27] Dorsa Sadigh, Katherine Driggs-Campbell, Alberto Puggelli, Wenchao Li, Victor Shia, Ruzena Bajcsy, Alberto Sangiovanni-Vincentelli, S Shankar Sastry, and Sanjit Seshia. 2014. Data-driven probabilistic modeling and verification of human driver behavior. In *2014 AAAI Spring Symposium Series*.

[28] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proc. of CVPR*.

[29] Mahadev Satyanarayanan, Victor Bahl, Ramón Caceres, and Nigel Davies. 2009. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing* (2009).

[30] Hamed Shah-Mansouri and Vincent WS Wong. 2018. Hierarchical fog-cloud computing for IoT systems: A computation offloading game. *IEEE Internet of Things Journal* 5, 4 (2018), 3246–3257.

[31] Haisheng Tan, Zhenhua Han, Xiang-Yang Li, and Francis CM Lau. 2017. Online job dispatching and scheduling in edge-clouds. In *Proc. of INFOCOM*.

[32] Liang Tong, Yong Li, and Wei Gao. 2016. A hierarchical edge cloud architecture for mobile computing. In *Proc. of INFOCOM*.

[33] Rahul Urgaonkar, Shiqiang Wang, Ting He, Murtaza Zafer, Kevin Chan, and Kin K Leung. 2015. Dynamic service migration and workload scheduling in edge-clouds. *Performance Evaluation* 91 (2015), 205–228.

[34] Francisco Vicente, Zehua Huang, Xuehan Xiong, Fernando De la Torre, Wende Zhang, and Dan Levi. 2015. Driver gaze tracking and eyes off the road detection system. *IEEE Transactions on Intelligent Transportation Systems* 16, 4 (2015), 2014–2027.

[35] Lin Wang, Lei Jiao, Ting He, Jun Li, and Max Mühlhäuser. 2018. Service entity placement for social virtual reality applications in edge computing. In *Proc. of INFOCOM*.

[36] Kathy Winter. 2017. *For Self-Driving Cars, There's Big Meaning Behind One Big Number: 4 Terabytes.* https://newsroom.intel.com/editorials/self-driving-cars-big-meaning-behind-one-number-4-terabytes/

[37] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. 2017. End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2174–2182.

[38] Xu Zhang, Siddhartha Sen, Daniar Kurniawan, Haryadi Gunawi, and Junchen Jiang. 2019. E2E: embracing user heterogeneity to improve quality of experience on the web. In *Proc. of ACM SIGCOMM*.