Online Job Dispatching and Scheduling in Edge-Clouds

Haisheng Tan*, Zhenhua Han*†, Xiang-Yang Li*, Francis C.M. Lau†
* University of Science and Technology of China (USTC), Hefei, China
† The University of Hong Kong (HKU), Hong Kong, China

***Abstract*—In edge-cloud computing, a set of edge servers are deployed near the mobile devices such that these devices can offload jobs to the servers with low latency. One fundamental and critical problem in edge-cloud systems is how to dispatch and schedule the jobs so that the job response time (defined as the interval between the release of a job and the arrival of the computation result at its device) is minimized. In this paper, we propose a general model for this problem, where the jobs are generated in arbitrary order and times at the mobile devices and offloaded to servers with both upload and download delays. Our goal is to minimize the total weighted response time over all the jobs. The weight is set based on how latency sensitive the job is. We derive the first online job dispatching and scheduling algorithm in edge-clouds, called `OnDisc`, which is *scalable* in the speed augmentation model; that is, `OnDisc` is $(1+\varepsilon)$-speed $O(1/\varepsilon)$-competitive for any constant $\varepsilon \in (0,1)$. Moreover, `OnDisc` can be easily implemented in distributed systems. Extensive simulations on a real-world data-trace from Google show that `OnDisc` can reduce the total weighted response time dramatically compared with heuristic algorithms.**

I. Introduction

With the development of cloud computing, more and more mobile applications can leverage the rich computing resources in cloud data centers [1]–[4]. Instead of relying on the resource-limited mobile devices, mobile applications can offload their computation-intensive jobs, such as image processing tasks, to remote clouds. Although such job offloading could significantly expand the capability of mobile devices, a long communication delay gets in the way inevitably since the remote clouds are usually far away from the mobile users. Such long latencies may severely downgrade the user experience especially for delay-sensitive applications. To mitigate the problem of long latencies, *edge-clouds*, also known as edge computing, fog computing, and cloudlets, have been proposed, which place a number of small scale servers at the network edge. Edge cloud servers can be reached by nearby mobile users via wireless connections. In this way, mobile devices can offload their tasks to edge clouds and receive the computing results with low network latency. However, because these specially deployed servers are of smaller scale than the remote ones, the resource and computation abilities of edge-clouds are relatively constrained when compared to the remote cloud data centers. Any particular edge-cloud might not be able to support a wide range of mobile applications. Therefore, edge-clouds are customarily backed by a remote cloud via the Internet so that they can offload some of their more demanding jobs to the remote cloud. Moreover, recent studies also proposed to connect multiple edge-clouds together (e.g., via a metropolitan area network) so as to enhance the services provided to the mobile users by sharing and balancing the workloads among the participating edge-clouds [5]–[7].

In an edge-cloud network, the fundamental and critical problem is when and where to offload a job from a mobile user. A mobile device can choose between the remote cloud and a nearby edge cloud to offload a job, which could be based on such considerations as the latency, the amount of computation and the resources required. This is a *dispatching* problem. On the other hand, a remote or edge cloud server has to determine which offloaded tasks should be served first, which is a *scheduling* problem. Concerning scheduling, most works on edge-clouds assumed a First-Come-First-Serve scheme (e.g. [5], [8], [9]). There were some studies on dispatching the jobs to achieve load-balancing [5], [8], [10]–[13]. However, most of them assumed that the releases of jobs follow some known stochastic process, so that they can make use of the statistical knowledge of the jobs to implement an efficient dispatch strategy. Actually, real-world mobile applications, the stochastic optimization based solution may perform badly due to the rapid changes and arbitrary releases of the jobs which deviate significantly from the assumed distribution. Therefore, an online dispatching and scheduling scheme without any assumption about the distribution of the job releases is desired in edge-cloud systems. Furthermore, in our study, the expected latency is the focus. Note however that in some mission-critical systems, a performance guarantee in the worst case is more important than the expected value.

In this paper, we study online algorithms for the job dispatching and scheduling problem in edge-cloud systems. The goal is to minimize the total job response time. The response time of a job is defined as the interval between the job's release in a mobile device and the time when it is finished and the result is received by the device. The interval can include the communication latency between the mobile device and the server, the waiting time and the processing time in the server. The online algorithm possess no future knowledge of the coming jobs. Moreover, to be compliant with practical scenarios, we assume the following general setting: 1) The jobs are released by the mobile devices in *arbitrary* order and times. 2) *Weighted response time* (WRT for short): we attach to the response time of each job a weight, to

Part of Zhenhua Han's work was done when visiting at USTC. Correspond with Z. Han at zhenhua@connect.hku.hk.

indicate its latency sensitivity; more latency-sensitive jobs will be assigned a larger weight so that they can be served with a higher priority. 3) *Unrelated machines model*: in most previous works, the processing time of a job in a server is calculated as the job size divided by the server's computation speed; then, although the machines[1] are not identical, the job processing times in different servers are related. In this work, we assume a general setting where each job has a machine-dependent processing time in each server, and there is no relationship among the job processing times in different servers. 4) *Upload and download delay*: for each pair of job j and server k, there is not only an upload delay for offloading jobs to the edge-cloud or remote server but also a download delay for the server to send back the computation results, denoted as $\Delta_{kj}^{\uparrow},\ \Delta_{kj}^{\downarrow} \geq 0$, respectively.

We use the *competitive ratio* as a metric to evaluate our online algorithm, defined as the largest possible ratio between the performance of the online algorithm and the offline optimal algorithm for any possible set of jobs. Therefore, it is a worst-case analysis of the online algorithm. Even without the upload and download delay, i.e., $\Delta_{kj}^{\uparrow} = \Delta_{kj}^{\downarrow} = 0, \forall k, j$, the job dispatching and scheduling problem can be proved to be without a bounded competitive ratio based on the results in [14]. Thus, during our theoretical analysis, we adopt the speed augmentation model [15], where the servers and the data transmission in the online algorithm are $(1+\varepsilon)$ times as fast as that in the optimal offline algorithm. Our contributions can be summarized as follows.

- We formulate the job dispatching and scheduling problem in edge-clouds under a general model as described in Sec III, and design an online algorithm to minimize the total WRT of all jobs.
- We propose an online *scalable* algorithm for the job dispatching and scheduling problem, called `OnDisc`, with a competitive ratio of $O(\frac{1}{\varepsilon})$ at $(1+\varepsilon)$-speed, where $\varepsilon \in (0,1)$ is a constant[2]. To the best of our knowledge, this is the first online algorithm for the job dispatching and scheduling problem in edge-clouds with a nontrivial competitive ratio. Our result also extends the theoretical machine scheduling problem (e.g., [14], [15]) by considering the job upload and download delays.
- For the model where mobile devices can choose to process their jobs themselves, we claim that `OnDisc` can still achieve the above competitive ratio by determining whether each job is to be processed locally at its device or offloaded to a server. We also discuss how to implement `OnDisc` in distributed systems.
- Based on a real workload trace from Google, we conduct extensive simulation to evaluate the performance of our online algorithm. The experimental results show that `OnDisc` can significantly reduce the WRT of the jobs when compared to heuristic algorithms. Our simulation results also confirm that it is practical to implement `OnDisc` in distributed systems.

The remainder of this paper is organized as follows. In Section II, we present the related works. In Section III, we give the formal definitions of the system model and the problem. We propose and analyze our online dispatching and scheduling algorithm in Section IV. In Section V, we evaluate our algorithm by simulations with a real-world data trace. We conclude the paper in Section VI.

II. Related Work

A. Cloud and Edge Computing

With cloud computing as the enabling technology, offloading heavy computation jobs to the remote cloud data centers has been studied for over a decade. CloneCloud [1] was proposed to use cloned virtual machine images in the cloud for mobile job offloading. ThinkAir [2] made an improvement by parallelizing the execution to enhance the power of mobile job offloading. COMET [16] adopted the distributed shared memory model to allow the application threads to migrate freely between the mobile device and the cloud server.

However, job offloading to the remote cloud suffers from heavy latency between the mobile devices and the remote cloud. Therefore, the concept of edge-clouds was proposed to provide nearby rich computing resources to the mobile users [17]. The computing resources at the edge-clouds are still limited. Thus there have been quite a lot of studies on the load sharing and balancing problem among the edge clouds and the remote clouds. The intuitive idea to offload the jobs to the nearest edge-clouds cannot work since it may lead to too many mobile devices competing for the limited computing resources of one edge-cloud [5], [8], [10]. Urgaonkar et al. [8] formulated the workload scheduling problem as a Markov Decision Process problem and adopted some Lyapunov optimization technique to solve this problem. Tong et al. [10] proposed a hierarchical architecture for the edge-clouds. They divided the edge-clouds into different levels according to the distance to the edge, so that the peak load at the edge-clouds can be offloaded to the higher tier edge-clouds. They also presented a heuristic algorithm to dispatch the workload within this hierarchical architecture. Based on the hierarchical architecture, they designed a heuristic algorithm to dispatch the jobs according to their loads at the edge-clouds. Jia et al. [5] pointed out that the load balancing among the edge-clouds can bring huge performance gain, much more than only processing the jobs at the edge-clouds in isolation. This model is useful in the metropolitan area networks where the edge-cloud servers can communicate with a low latency. Most of the above load balancing schemes in the edge-clouds are based on stochastic optimization, for which they assume that the job release process follows a certain distribution. However, in practice, the jobs released from mobile device can be in arbitrary order and times. Therefore, online algorithms without any assumption on the job release distribution are desired.

Besides designing algorithms for a fixed edge-cloud configuration, there are a number of works studying reconfiguration

[1] In this paper, we use "machine" and "server" interchangeably.

[2] An online algorithm is called *scalable* if it is $O(1)$-competitive when given the smallest amount of computing resource over the offline algorithm.

on edge-clouds. Amble et al. [12] proposed a reconfiguration method by assuming that the request arrival is an independent and identically distributed (i.i.d.) process, while Wang et al. [13] considered a Markov process. Hou et al. [18] derived an online algorithm with competitive analysis, where they showed that their online algorithm can even perform better than the stochastic optimization based approaches, especially in the worst case.

B. Classic Online Machine Scheduling Problem

The problem of job dispatching and scheduling in edge-clouds has some similarities to the classic online machine scheduling problem on unrelated machines. In the classic problem, a list of jobs are released in an online manner, i.e., a job's information is unknown before its release. Each job can have different and unrelated processing times on different machines. A machine processes at most one job at one time. We need to decide how to dispatch the jobs and how to schedule the jobs at each machine to minimize the total (weighted) response time (also known as the flow time in the original papers cited here), which is the sum of the waiting and processing times of the jobs. It has been shown that no online algorithm can have a bounded competitive ratio, by Garg and Kumar [14]. Chadha et al. [15] proposed the speed augmentation model to analyze this kind of online algorithms, where an online algorithm can have ε-fraction more speed in a machine than one that runs an offline algorithm. They also derived a greedy algorithm with a competitive ratio of $O(\frac{1}{\varepsilon^2})$. Im and Moseley [19] improved the competitive ratio to $O(\frac{k}{\varepsilon^{2+2/k}})$ for the l_k-norm of total WRT. Anand et al. [20] further achieved a competitive ratio of $O(\frac{1}{\varepsilon})$ for the l_1-norm and $O(\frac{k}{\varepsilon^{2+1/k}})$ for the l_k-norm of the total WRT.

However, the above results cannot be applied to our job dispatching and scheduling problem in edge-clouds, because they assumed that the jobs can arrive at the machines instantly after their releases, while there is in fact a significant latency for the transmission between the mobile devices and the servers, especially when the servers are at the remote clouds. With the presence of communication latencies, at release time, a job does not know what might happen before its actual arrival at a machine. For example, after a mobile device decides to dispatch a job j to an idle edge-cloud server, it is possible that a lot of jobs from other devices are dispatched to the same server, which might even arrive before j. In this case, the job j will end up suffering a long waiting time due to resource contention on the server. Thus, it might be more difficult to design online algorithms for the job dispatching and scheduling problem in edge-clouds. We wish this work could inspire further studies on the classic machine scheduling problem with job transmission latency.

III. System Model

We consider an edge-cloud system with K heterogenous edge-cloud servers, $\mathcal{K} = \{s_1, s_2, ..., s_K\}$, each of which is configured to serve a set of mobile applications. There are a set $\mathcal{J} = \{j_1, j_2, ...\}$ of indivisible jobs (if a job can be divided, we will treat each portion as a new job) released by the mobile devices in arbitrary order and times. We set r_j as the release time of the job j. A mobile device will dispatch a job to an edge server or the remote clouds immediately after its release[3]. We do not allow the servers to migrate a job to other servers after the dispatching to avoid migration overhead.

A server can execute at most one job at a time preemptively; that is, the server can switch to another job and resume the current job later. Thus, each server should carefully schedule its unfinished jobs. There is an upload delay $\Delta_{kj}^{\uparrow}$ for job j to be dispatched to s_k. Thus, s_k cannot start processing the job j until time $r_j + \Delta_{kj}^{\uparrow}$. The processing time for job j at s_k is denoted as p_{kj}. Also, there is a download delay $\Delta_{kj}^{\downarrow}$ which is the latency for the mobile device to download the result of job j after its computation at s_k. Note that the uploading and downloading of a job do not consume any computing resource of the servers, so a server can process a job while transmitting other jobs at the same time. Because the remote cloud servers can be modeled as edge-cloud servers with long transmission delay and more powerful processing capability, we do not explicitly differentiate between the edge-cloud servers and the remote cloud servers here.

For a job j, we use the term "release" to indicate its generation by a mobile device at time r_j, and "arrival" as the completion of its dispatching and uploading to s_k at $r_j + \Delta_{kj}^{\uparrow}$. Moreover, "computation completed" indicates its processing is finished by an s_k, and the term "fully completed" means the result of the job has reached the mobile device. Each job j has a weight w_{kj} when dispatched to s_k, where $w_{kj} = +\infty$ if the server cannot process this job; otherwise, w_{kj} is a parameter to indicate how sensitive it is to the delay. Recall that the response time of job k processed at s_k is the duration between its release and the time when it is fully completed. Then, its WRT is w_{kj} times the response time. We assume that for any job, there is at least one server that can process it.

We study the online job dispatching and scheduling problem in edge-cloud systems. The information of any job j is unknown until it is released by a mobile device, including its upload delay $\Delta_{kj}^{\uparrow}$, its download delay $\Delta_{kj}^{\downarrow}$, its job processing time p_{kj} and its weight w_{kj} at server s_k. Our goal is to minimize the total WRT of all the jobs. Recall that it has been proved there is no online algorithm with a bounded competitive ratio for this problem, even when the upload/download delays are set to zero. Therefore, we evaluate the performance of our online algorithm by its competitive ratio in the speed augmentation model, where the server and data transmission in the online algorithm can be $1 + \varepsilon$ times as fast as that in the offline algorithm. Formally, for a constant c, we define c-competitive in the speed augmentation model as:

Definition 1. *An online algorithm η is c-competitive with speed $(1 + \epsilon)$, if $\frac{\mathcal{F}^{\eta}}{OPT_{offline}} \leq c$ for every possible set of jobs,*

[3]Alternatively, the mobile device can offload its job to a nearby server which will determine instantly whether to process this job or to further dispatch it to other servers. In our problem, this is equivalent to the mobile device itself handling the dispatching.

where $\mathcal{F}^{\eta}$ and $OPT_{offline}$ denote the performance of the online algorithm η and the optimal offline algorithm, respectively.

IV. Online Dispatching and Scheduling

With the general model described above, we next propose our online dispatching and scheduling algorithm and present its competitive analysis. We also discuss how to extend the algorithm to suit more general settings, and how to apply it in distributed systems.

A. Online Algorithm

The algorithm is composed of two parts which are driven by two policies respectively: *the dispatching policy* and *the scheduling policy*. All the mobile devices follow the same dispatching policy to choose servers to offload their jobs, and all servers adopt the same scheduling policy to determine the processing order of their unfinished jobs.

• **Scheduling Policy:** Let $d_{kj} \triangleq w_{kj}/p_{kj}$ denote the weight *density* of job j on server s_k, which is its weight divided by its processing time. Set $p_j(t)$ as the remaining processing time of job j at time t. Then, we define the residual density of job j at time t as $d_j(t) \triangleq w_{ij}/p_j(t)$. Let $\mathcal{A}_k(t)$ denote the jobs at s_k, which have arrived but whose computations have not finished. Each server follows the Highest Residual Density First (HRDF) rule to schedule its unfinished jobs, i.e., s_k processes the job $j' = \text{argmax}_{j\in\mathcal{A}_k(t)}\, d_j(t)$ at time t. When there is a tie, the job arrives earlier has the higher priority.
Note that, on one server, if a job j_1 has a higher density than another job j_2 at some time t, j_1 will always have the higher density, i.e., if $\exists t,\ d_{j_1}(t) > d_{j_2}(t)$, then $d_{j_1}(t') > d_{j_2}(t'),\ \forall t' \geq t$. This is a nice property as it avoids frequent switches of jobs being processed on a server. If all jobs have a unique weight, the scheduling policy would be the same as the SRPT (Shortest Remaining Processing Time) policy [21].
• **Dispatching Policy:** When $t = r_j$, job j is released by a mobile device, and needs to be dispatched to a server instantly. Our policy is to greedily dispatch a job to the server which brings the least increase to the total WRT. Set s_j^* as the server to which job j is dispatched. Let $d_{kj}(t')$ be the residual density of job j at time t' if it is dispatched to s_k. Set t_{kj}^* as the computation completion time of j by assuming that no new job will be dispatched to s_k in the future. t_{kj}^* can be calculated by simulating the scheduling policy at server s_k.
If job j is dispatched to s_k, the increase of the total WRT is composed of three parts:

1) the weighted waiting time of j due to the other jobs at s_k with larger residual density than j (called Type-I jobs to j);
2) the weighted processing time and transmission delays of j;
3) extra weighted waiting time brought by j to the jobs with smaller residual density than j (called Type-II jobs to j).

Note that for job j, its Type-I and Type-II jobs may not only be the jobs that have already arrived at s_k, but also the jobs that have been released by time t and to be dispatched to s_k but not yet arrived. We define a set $\mathcal{A}_{kj}^{\dagger}(t', t)$ as follows:

$$\mathcal{A}_{kj}^{\dagger}(t',t) = \begin{cases} \{j' \mid s_{j'}^* = k, r_{j'} \leq t, r_{j'} + \Delta_{kj'}^{\uparrow} \leq t', \\ \qquad p_{j'}(t') > 0\}, & t' = t + \Delta_{kj}^{\uparrow} \\ \{j' \mid s_{j'}^* = k, r_{j'} \leq t, \\ \qquad r_{j'} + \Delta_{kj'}^{\uparrow} = t'\}, & t' > t + \Delta_{kj}^{\uparrow}, \end{cases}$$

where $p_{j'}(t')$ is the remaining processing time of job j' at time t' by assuming there will be no job dispatched to s_k after time t. That is to say, when $t' = t + \Delta_{kj}^{\uparrow}$, $\mathcal{A}_{kj}^{\dagger}(t', t)$ is the set of jobs which have arrived at s_k but whose computation has not been finished; when $t' > t + \Delta_{kj}^{\uparrow}$, $\mathcal{A}_{kj}^{\dagger}(t', t)$ is the set of jobs that have been dispatched to s_k by time t and will arrive exactly at time t'. Obliviously, the Type-I and Type-II jobs to job j are all in $\mathcal{A}_{kj}^{\dagger}(t', t)$.
Let $\mathcal{T}_k(t)$ denote the set of times when there is at least one job, which has been released by t, arrives at s_k [4]. Define $\mathcal{A}_{kj}^{1}(t)$ as a set of the combinations of job j' in Type-I for job j and a time t', which is given by $\mathcal{A}_{kj}^{1}(t) = \{(j', t') | t' \in \mathcal{T}_k(t) : t_{kj}^* > t' \geq t + \Delta_{kj}^{\uparrow}, j' \in \mathcal{A}_{kj}^{\dagger}(t', t) : d_{j'}(t') \geq d_{kj}(t')\}$. Similarly, we define $\mathcal{A}_{kj}^{2}(t)$ as a set of the combinations of Type-II jobs and the time, which is given by $\mathcal{A}_{kj}^{2}(t) = \{(j', t') | t' \in \mathcal{T}_k(t) : t_{kj}^* > t' \geq t + \Delta_{kj}^{\uparrow}, j' \in \mathcal{A}_{kj}^{\dagger}(t', t) : d_{j'}(t') < d_{kj}(t')\}$.
Based on the above notations, when job j is released at time t, we can now compute the increase of the total WRT, denoted as $Q_{kj}(t)$, if it is dispatched to s_k.

$$Q_{kj}(t) = \frac{1}{1+\varepsilon}\Big\{ w_{kj} \sum_{(j',t')\in\mathcal{A}_{kj}^{1}(t)} p_{j'}(t') + w_{kj}(p_{kj} + \Delta_{kj}^{\uparrow} + \Delta_{kj}^{\downarrow}) + \sum_{(j',t')\in\mathcal{A}_{kj}^{2}(t)} p_{kj}(t') w_{kj'} \Big\}, \quad (1)$$

The first, second and third terms inside the brackets of Eqn. (1) denote Parts 1), 2) and 3) of the increase of the total WRT, respectively. $\frac{1}{1+\varepsilon}$ is the speed augmentation factor. Our dispatching policy is to assign the job j to s_k which minimizes $Q_{kj}(t)$, i.e., $s_j^* = \text{argmin}_{s_k\in\mathcal{K}} Q_{kj}(t)$.
Note that the following two types of jobs will not be included in the calculation of Eqn. (1), since they will never be affected by or affect the processing of job j: the jobs whose computation completes before time $t + \Delta_{kj}^{\uparrow}$ (when job j arrives at s_k); and the ones which arrive after time t_{kj}^* (when job j's computation completes).

Example 1. *Fig. 1 shows an example of calculating the increase of the total WRT for job* 0 *which is released at time t. When Job 0 is released and dispatched, there are also Jobs 1–5 dispatched to this server. After its arrival, Job* 0 *needs to wait until the Type-I jobs (Jobs 1 and 2) complete their computation. Also, Job 0 will introduce more waiting time to the Type-II jobs (Job 3) when Job 3 arrives at the server. In addition, Job 4 and Job 5 would not influence the calculation*

[4]Note that $\mathcal{T}_k(t)$ may contain a time larger than t, because there may be some jobs that have been dispatched to s_k but have not arrived by t.

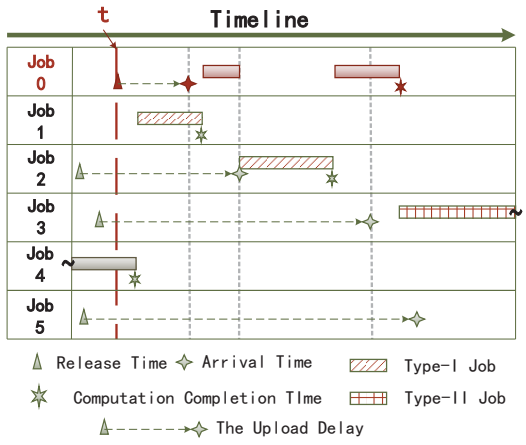


Fig. 1. Calculation of the increase of the total WRT for job j_0 released at time t. The download delay is not shown in this figure.

because Job 4's computation completes before the arrival of Job 0, and Job 5 arrives after the computation of Job 0.

Based on the dispatching and scheduling policies, we summarize our online algorithm `OnDisc` as Algorithm 1.

Algorithm 1: `OnDisc`

1 **Job Dispatching:** when a job j is released at $t = r_j$, it is dispatched to the server $s_j^* = \text{argmin}_{s_k \in \mathcal{K}} Q_{kj}(t)$;
2 **Job Scheduling on a Server:** at any time t', the server s_k processes the job $j^* = \text{argmin}_{j \in \mathcal{A}_k(t')} d_j(t')$;

B. The Competitive Analysis

Let $\mathcal{F}$ denote the total WRT obtained by our online algorithm `OnDisc`. In order to analyze the competitive ratio, we have to find a lower bound of the total WRT for our job dispatching and scheduling problem in the offline mode, i.e., all the information of the jobs are known beforehand. Here, we can the *offline* job dispatching and scheduling problem as *the original problem*. We mainly adopt the dual fitting technique [20], and the general idea is as follows:

- We first derive an LP (Linear Programming) relaxation of the original problem, and prove that the LP-relaxation provides a lower bound.
- Then, we write out the dual LP to the LP relaxation and construct a feasible solution to the dual problem. A feasible solution to the dual LP is always a lower bound of the primal LP, and hence also a lower bound of our original problem.
- By showing the relationship between $\mathcal{F}$ and the objective in the dual LP, we can finally derive the competitive ratio of our online algorithm `OnDisc`.

1) the primal LP relaxation: We divide the time into slots. A server can allocate different fractions of a time slot to process different jobs. However, recall that a server can only process one job at most at a specific time point. Set x_{kjt} as the fraction of time slot $[t, t+1]$ used by server s_k to process job j after j's arrival, i.e., $t \geq r_j + \Delta_{kj}^{\uparrow}$. For a job j, we tailor-make a parameter $\Lambda_j(\mathbf{x})$ defined as

$$\Lambda_j(\mathbf{x}) \triangleq \sum_{k,t} w_{kj} \cdot x_{kjt} \cdot \left(\frac{t-(r_j+\Delta_{kj}^{\uparrow})}{p_{kj}} + \frac{1}{2}(1 + \frac{\Delta_{kj}^{\uparrow}+\Delta_{kj}^{\downarrow}}{p_{kj}}) \right), \quad (2)$$

where $\mathbf{x}$ is the collection of all x_{kjt}. The primal LP relaxation of the original problem is shown in Problem 1.

Problem 1 (The Primal LP Relaxation)**.**

$$\min \quad \sum_j \Lambda_j(\mathbf{x}) \quad (3)$$

$$s.t. \quad \sum_{k,t} \frac{x_{kjt}}{p_{kj}} \geq 1 \quad \forall j, \quad (4)$$

$$\sum_j x_{kjt} \leq 1 \quad \forall k, t, \quad (5)$$

$$x_{kjt} \geq 0 \quad \forall j, k, t \geq r_j + \Delta_{kj}^{\uparrow}. \quad (6)$$

The constraint (4) means any job j must be completely processed. Note that here a job can be divided into fractions and allocated to multiple servers, and even the different fractions can be processed simultaneously in different servers. Constraint (5) refers to the fact that each server can process at most one job at a specific time point. Therefore, the constraints in Problem 1 are a relaxation of the original problem where the jobs can only be dispatched and executed at one server. Let S be any feasible dispatching and scheduling strategy for the original problem. It is clear that we can uniquely translate S to a feasible solution, denoted as $\mathbf{x}_S$, to Problem 1.

We have the following lemma that claims Problem 1 gives a lower bound for the original offline dispatching and scheduling problem in edge-clouds.

Lemma 1. *The total WRT of any feasible strategy S for the original problem is at least $\sum_j \Lambda_j(\mathbf{x}_S)$.*

Proof: For job j, we assume in strategy S it is dispatched to s_k. Denote t_c as the computation completion time of j, so that $x_{kjt''} = 0, \ \forall t'' > t_c$. $\Lambda_j(\mathbf{x}_S)$ is maximized when job j is processed from $t_c - p_{kj}$ to t_c. Thus we have

$$\Lambda_j(\mathbf{x}_S) \leq w_{kj} \sum_{t'=1}^{p_{kj}} \left(\frac{t_c-(r_j+\Delta_{kj}^{\uparrow})-t'}{p_{kj}} + (\frac{1}{2} + \frac{\Delta_{kj}^{\uparrow}+\Delta_{kj}^{\downarrow}}{2 \cdot p_{kj}}) \right)$$

$$= w_{kj}(t_c - r_j - \frac{\Delta_{kj}^{\uparrow}}{2} + \frac{\Delta_{kj}^{\downarrow}}{2}) \leq w_{kj}(t_c - r_j + \Delta_{kj}^{\downarrow}). \quad (7)$$

The rightmost term in Eqn. (7) is the WRT of job j in strategy S. Therefore, any feasible solution S to the original problem has a total WRT at least $\sum_j \Lambda_j(\mathbf{x}_S)$. ■

2) the dual LP: The dual LP of the primal LP relaxation in Problem 1 is given as

Problem 2 (The Dual LP to Problem 1)**.**

$$\max \quad \sum_j \alpha_j - \sum_{k,t} \beta_{kt} \quad (8)$$

$$s.t. \ \frac{\alpha_j}{p_{kj}} - \beta_{kt} \leq d_{kj}(t - (r_j + \Delta_{kj}^{\uparrow})) + \frac{w_{kj}}{2}(1 + \frac{\Delta_{kj}^{\uparrow}+\Delta_{kj}^{\downarrow}}{p_{kj}}) \quad \forall k, j, t \geq r_j + \Delta_{kj}^{\uparrow}, \quad (9)$$

$$\alpha_j \geq 0, \beta_{kt} \geq 0 \quad \forall k, j, t, \quad (10)$$

where α_j and β_{kt} are the dual variables.

Recall that $\mathcal{F}$ denotes the total WRT in our online algorithm `OnDisc`. Our goal is to achieve what is stated in the following theorem.

Theorem 1. *There exists a feasible solution α_j and β_{kt} to Problem 2, such that the objective value (i.e., $\sum_j \alpha_j - \sum_{kt} \beta_{kt}$) is $\Omega(\varepsilon \cdot \mathcal{F})$.*

Proof: We need to find a set of values for the dual variables α_j and β_{kt} to satisfy Theorem 1. We define α_j^* as the increase of the total WRT by job j in `OnDisc`, i.e., $\alpha_j^* = \min_k Q_{kj}(r_j)$. Let $\mathcal{B}_k(t)$ denote the set of unfinished jobs which have been dispatched to s_k at time t [5]. We define β_{kt}^* to equal to $\frac{1}{1+\varepsilon}$ times the total weights of the jobs in $\mathcal{B}_k(t)$, i.e., $\beta_{kt}^* = \frac{1}{1+\varepsilon}\sum_{j\in\mathcal{B}_k(t)} w_{kj}$.

It is obvious that $\sum_{k,t}\beta_{kt}^*$ is exactly equal to $\frac{\mathcal{F}}{1+\varepsilon}$, and $\sum_j \alpha_j^*$ is equal to $\mathcal{F}$. Therefore, we have $\sum_j \alpha_j^* - \sum_{k,t}\beta_{kt}^* = \frac{\epsilon}{1+\epsilon}\cdot\mathcal{F}$.

Lemma 2. *By setting $\alpha_j = \alpha_j^*/2$ and $\beta_{kt} = \beta_{kt}^*/2$, we get a feasible solution to Problem 2.*

Proof: Fix a server k and a job j. Let t denote the release time of j and $t^\dagger \geq t + \Delta_{kj}^{\uparrow}$. We need to prove

$$\frac{\alpha_j^*}{p_{kj}} - \beta_{kt^\dagger}^* \leq d_{kj}(t^\dagger - (t+\Delta_{kj}^{\uparrow})) + \frac{w_{kj}}{2}(1+\frac{\Delta_{kj}^{\uparrow}+\Delta_{kj}^{\downarrow}}{p_{kj}}) \quad (11)$$

All terms in (11) will not be affected by the jobs being released after t, except for $\beta_{kt^\dagger}^*$ by increasing its value (as there will be more jobs in the set $\mathcal{B}_k(t^\dagger)$). Therefore, We can assume there is no new job released after job j.

We suppose s_k will process job j_s at time $t^\dagger$. Let t_s denote the corresponding time of j_s appearing in $\mathcal{A}_{kj}^1(t)\bigcup A_{kj}^2(t)$, i.e., $(j_s, t_s) \in \mathcal{A}_{kj}^1(t)\bigcup A_{kj}^2(t)$. There are two possible cases: (1) $(j_s,t_s)\in\mathcal{A}_{kj}^1(t)$, (2) $(j_s,t_s)\in\mathcal{A}_{kj}^2(t)$, or j_s never appears in $\mathcal{A}_{kj}^1(t)\bigcup\mathcal{A}_{kj}^2(t)$. In both cases, we can achieve:

$$\begin{aligned}\frac{(1+\varepsilon)\alpha_j^*}{p_{kj}} &< d_{kj}(t^\dagger - (t+\Delta_{kj}^{\uparrow}))(1+\varepsilon) \\ &\quad + w_{kj}(1+\frac{\Delta_{kj}^{\uparrow}+\Delta_{kj}^{\downarrow}}{p_{kj}}) + \beta_{kt^\dagger}^*(1+\varepsilon) \quad (12)\end{aligned}$$

We skip the details due to limited space. By dividing both sides of Eqn. (12) by $2(1+\varepsilon)$, we have

$$\frac{\alpha_j^*}{2\cdot p_{kj}} \leq \frac{d_{kj}(t^\dagger - (t+\Delta_{kj}^{\uparrow}))}{2} + \frac{w_{kj}(1+\frac{\Delta_{kj}^{\uparrow}+\Delta_{kj}^{\downarrow}}{p_{kj}})}{2(1+\varepsilon)} + \frac{\beta_{kt^\dagger}^*}{2}$$

$$\Rightarrow \frac{\alpha_j^*}{2\cdot p_{kj}} - \frac{\beta_{kt^\dagger}^*}{2} \leq d_{kj}(t^\dagger - (t+\Delta_{kj}^{\uparrow})) + \frac{w_{kj}}{2}(1+\frac{\Delta_{kj}^{\uparrow}+\Delta_{kj}^{\downarrow}}{p_{kj}})$$

Therefore, it can be determined that $\alpha_j^*/2$ and $\beta_{kt}^*/2$ satisfy the inequality (11) in both Cases 1 and 2.

[5]It includes the unfinished jobs that have arrived at s_k, the jobs that have not arrived but have been released and dispatched to s_k, and the jobs that have been completely processed but the results have not been fully downloaded.

Moreover, if all the jobs dispatched to s_k have been completed (s_k is idle), the inequality (11) still holds due to the fact that $\beta_{kt^\dagger}^* \geq 0$ and $\frac{w_{kj}}{2}(t^\dagger - (t+\Delta_{kj}^{\uparrow}) + \Delta_{kj}^{\downarrow}) \geq \frac{\alpha_j^*}{2}$ in this case. This completes the proof of this lemma. ■

Thus, we get that $\alpha_j = \alpha_j^*/2$ and $\beta_{kt} = \beta_{kt}^*/2$ are the feasible solution to Problem 2. And the objective function $\sum_j \alpha_j - \sum_{kt}\beta_{kt} = \frac{1}{2}(\sum_j \alpha_j^* - \sum_{k,t}\beta_{kt}^*) = \frac{\varepsilon}{2(1+\varepsilon)}\mathcal{F} = \Omega(\varepsilon)\cdot\mathcal{F}$, when $\varepsilon \in (0,1)$ is constant. This completes the proof of Theorem 1. ■

Recall that the objective value of Problem 2 is a lower bound of the total WRT in the offline dispatching and scheduling problem. We have

Corollary 1. *The online algorithm `OnDisc` is $O(\frac{1}{\varepsilon})$-competitive with $(1+\varepsilon)$-speed augmentation.*

C. Discussion

1) Offloading or Locally Processing a Job: By now, we have only studied the dispatching and scheduling strategy for the jobs to be offloaded to the edge-clouds and the remote clouds. Some previous works (e.g., [22], [23]) also allowed the jobs to be executed locally at the mobile device and studied the problem of whether to offload a job or not. Our algorithm, `OnDisc`, can be extended to suit the same model and make online decision as to whether to offload a job or not. For a mobile device m, we can regard it as a less powerful edge-cloud server with zero upload/download delay to the jobs released by itself. Moreover, the jobs released by the other mobile devices are set to have an infinitely large weight on m so that these jobs will never be dispatched to m.

Note that by setting finite weights of jobs released by one mobile device at other devices, we can get a more general model in which the mobile devices can offload tasks onto one another. Clearly, our algorithm can also work in this model.

2) No Resource Contention at Remote Clouds: In our system model, remote cloud servers have no essential difference from edge-cloud servers, except that they have larger upload/download delays and more powerful processing capability. Resource contention may still exist and hence there could be a waiting time for the jobs in the remote clouds. However, in some other models (e.g., [8], [10], [24]), when a job is dispatched to a remote cloud, a new server can be activated to process the job if there is no available idle server that is already activated. Therefore, there will be no waiting time for the jobs offloaded to the remote clouds. Our online algorithm `OnDisc` can also be modified to work with this model, where there will be no scheduling in the remote clouds and we need not consider the waiting time at remote clouds during dispatching. This modification will change our competitive analysis a little: i.e., a new LP relaxation is required. Due to the limited space, we omit the details here.

3) Distributed Implementation: In order to get efficient dispatching and scheduling algorithms in edge-clouds, most previous works (e.g., [5], [8], [9], [13]) required a central controller, which collects the states of all servers in real-time. This is difficult to implement in practice when edge-cloud servers are extensively deployed. It will lead to system

hiccup if the controller is attacked or experiences a hardware failure. Therefore, a distributed dispatching and scheduling algorithm is desired. Thanks to the simplicity of our online algorithm, OnDisc, the algorithm can be easily implemented in a distributed fashion. Firstly, our HRDF-based scheduling can work locally at each server. We only need to consider how to implement the dispatching policy distributedly, which can be achieved by following steps:

When a job j is released by a mobile device at time t,

- **Step 1**: the device sends the information of job j to each server, including the upload/download delay, the processing time, and the weight.
- **Step 2**: each server s_k calculates $Q_{kj}(t)$ in Eqn. 1 and replies the value to the mobile device.
- **Step 3**: after receiving all $Q_{kj}(t)$ from servers, the mobile device dispatches j to the server with the smallest Q_{kj}.

In Steps 1 and 2, in order to compute $Q_{kj}(t)$, the mobile device needs to exchange information with all the servers. The communication overhead with the edge-clouds nearby is small. However, the overhead for communication with remote clouds is so large that the job latency will be increased dramatically. The following discussion highlights how we can try to overcome this overhead. In the model without resource contention at remote clouds, when the server s_k is in the remote clouds, there will be no waiting time for job j, and $Q_{kj}(t)$ becomes the sum of the processing time and the upload/download delay, all of which can be known by the mobile device beforehand. Therefore, the value of $Q_{kj}(t)$ can be calculated locally at the mobile device without communication with the remote clouds. Moreover, for the model that resource contention might exist in remote clouds, we can still apply our online algorithm OnDisc by only exchanging information with edge-clouds nearby. Because the remote clouds have much more servers than the edge-clouds, the waiting time in the remote clouds is relatively low so that it can be ignored approximately. Our simulations in Sec V-C2 validate that with enough servers, OnDisc can still achieve a good performance when ignoring the contention in the remote clouds. Therefore, during the distributed implementation of OnDisc, we need only to exchange information with edge-clouds in Steps 1 and 2, and the communication overhead becomes affordable.

V. Performance Evaluation

In this section, we evaluate the performance of the proposed online algorithm OnDisc by extensive simulations and using a real-world data trace, and we compare it with two heuristic baselines.

A. Experiment Settings

We use the data set from the Google cluster [25] to obtain the release time, the processing time and the weights ("priority" in the data trace) of the jobs. The data trace contains more than 120000 jobs, among which we randomly choose 50000 jobs and group them into 10 non-overlapping sets. All the experiments here are conducted on the 10 job sets independently, and all the performance numbers are reported in average. If not specified explicitly, the upload/download delay to/from the edge-clouds and the remote clouds are set in the range of $[0.01, 0.03]$ seconds and $[0.2, 0.4]$ seconds, respectively. The processing time in a remote cloud server is set as ~ 0.75 times (in average value) of that in the edge-cloud servers.

B. Two Heuristic Baselines

In addition to OnDisc, we also implement two heuristic baselines, called "*Nearest*" and "*Selfish*". All the algorithms employ the dispatching and scheduling procedures to reduce the total WRT. The dispatching of the two baselines is designed as

- *Nearest:* after a job is released, dispatch the job to the server with the smallest upload and download delay.
- *Selfish:* a job is dispatched to the server with the earliest weighted completion time for this job assuming that there will be no job dispatched to this server in the future.

Nearest is the most friendly approach to distributed systems, as the mobile device only needs to communicate with the nearest edge-clouds [26]. Most of the works (e.g., [5], [27]) on edge-clouds adopted *Nearest* as the job dispatching policy. *Selfish* is close to the practical scenarios where each job only cares about their own performance [28]. In the figures, we will use O, N and S to denote OnDisc, *Nearest* and *Selfish*, respectively.

As for the scheduling policy, we adopt the Highest Residual Density First (HRDF) in OnDisc. Another popular scheduling policy is the First-Come-First-Serve (FCFS), which is commonly used by the approaches based on queuing theory (e.g., [5], [8], [9]). We will combine both scheduling policies with the dispatching polices and analyze the performance later.

If we use HRDF in *Selfish*, job j will be offloaded to the server $k = \text{argmin}_{k'} Q^{\text{selfish}}_{k'j}(t)$, where Q^{selfish}_{kj} is defined as:

$$Q^{\text{selfish}}_{kj}(t) = \frac{1}{1+\varepsilon}\Big\{ w_{kj} \sum_{(j',t') \in \mathcal{A}^1_{kj}(t)} p_{j'}(t') + w_{kj}(p_{kj} + \Delta^{\uparrow}_{kj} + \Delta^{\downarrow}_{kj}) \Big\}. \quad (13)$$

Note that Eqn. (13) is obtained from Eqn. (1) by dropping the Type-II jobs. That is, in *Selfish*, one job only cares about the performance of itself, but not the influence it brings to the other jobs, i.e., the extra weighted waiting time to other jobs with a smaller residual density.

C. Simulation Results

In this part, we present the simulation results on the impact of the scheduling policies (FCFS and HRDF), the remote clouds and the edge-clouds on the performance of the algorithms. In all cases, our online algorithm OnDisc outperforms the other baseline algorithms.

1) FCFS v.s. HRDF: Table I shows the average weighted response time over the combinations of the dispatching and scheduling policies. We configure 10 edge-servers with totally randomly chosen 1000 jobs and there is no remote cloud server. We can see HRDF always outperforms FCFS. Since FCFS is non-preemptive, the performance gain of HRDF can be regarded as the advantage of preemption in scheduling.

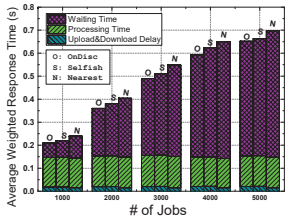
TABLE I
PERFORMANCE WITH FCFS OR HRDF

Avg WRT: Scheduling / Dispatching	FCFS	HRDF
Nearest	0.243s	**0.235s**
Selfish	0.177s	**0.158s**
Greedy in `OnDisc`	0.154s	**0.153s**

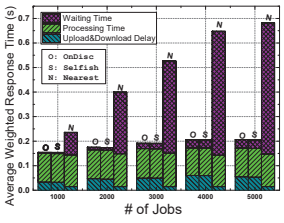
Interestingly, the performance gain of HRDF in *Selfish* is much higher than that in *Nearest* and `OnDisc`. With HRDF, *Selfish* can even achieve similar performance to `OnDisc`. It can be explained as: when all jobs choose to be selfish in dispatching, a more rational scheduling policy (such as HRDF) can alleviate the impact of selfishness on the overall performance of the system.

Since HRDF always outperforms FCFS, we adopt HRDF in scheduling for all the algorithms in the rest of this paper.

2) The Impact of Remote Clouds: In the subsection, we analyze how the remote clouds impact the WRT in edge-cloud systems. Because *Nearest* only dispatches the jobs to the nearest edge-cloud server, its performance will never be impacted by the remote cloud servers.



(a) No remote cloud server



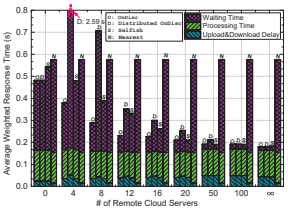
(b) 100 remote cloud servers with resource contention

Fig. 2. The average weighted flow time with different number of jobs.

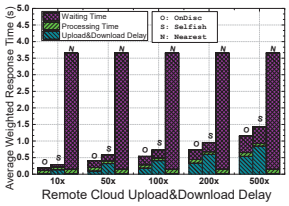
Existence of Remote Clouds: Fig. 2 demonstrates the performance with and without the remote cloud servers when the number of jobs increases from 1000 to 5000. In Fig. 2(a), when there is no remote cloud, the average WRT severely increases when more jobs come. Specifically, the job waiting time becomes much longer and plays the most important role in the response time. It is easily understood since more jobs will incur more resource contention in edge-clouds. With powerful remote cloud servers to offload the jobs, Fig. 2(b) shows that the average WRT of `OnDisc` and *Selfish* dramatically decreases, although longer upload/download delays are introduced.

Number of Remote Cloud Servers: Fig. 3 illustrates the weighted average response time affected by the number of servers in remote clouds. We set the number of edge-cloud servers to 10. Here, we also evaluate the distributed implementation of our algorithm `OnDisc` in Sec IV-C3 (denoted as D in the figures), which always ignores the resource contention in the remote clouds. Fig. 3(a) shows the case with 3000 jobs. As 3000 jobs would exceed the processing capacity of edge-cloud servers, the remote cloud servers can significantly reduce the average WRT. We can see that with more servers in the remote clouds, the WRT of all the algorithms decreases except for *Nearest* as it never adopts any remote cloud.

In Fig. 3(a), when the number of remote cloud servers is small, our distributed `OnDisc` performs badly since there is actually severe resource contention (and hence long job waiting time) in the remote clouds which should not be ignored. With the increase of the number of remote servers, the performance of distributed `OnDisc` gets closer and closer to `OnDisc`. In the last column with infinite number of remote servers, there is indeed no resource contention in the remote clouds (i.e. the model mentioned in Sec IV-C2); then, the distributed `OnDisc` achieves the same performance as `OnDisc`. This simulation result validates that our online algorithm `OnDisc` can be implemented distributedly with small communication overhead when there are enough computation resources in the remote clouds as discussed in Sec IV-C3.



(a) Impact of the number of remote cloud servers



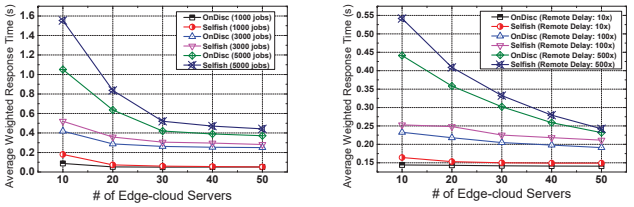
(b) Impact of the upload&download delay of the remote cloud servers

Fig. 3. The impact of the remote cloud server on the average WRT.

Upload/Download Delay to the Remote Clouds: Fig. 3(b) shows the impact of the upload and download delay on the remote clouds. In this experiment, there is one remote cloud server with no resource contention and we evaluate the algorithms by changing the upload and download delay to the remote server as a multiple (from 10 to 500 times) of the average upload and download delay of the edge-cloud servers. We can find that `OnDisc` and *Selfish* perform worse with larger delays. Note that the job waiting time also increases with larger remote cloud upload/download delay. The reason is that more jobs will be dispatched to the edge-cloud servers when the delay to the remote clouds becomes larger.

3) The Impact of Edge-cloud Servers: When designing an edge-cloud system, a natural problem arises: how many edge-cloud servers are enough? In other words, we need to figure out the factors affecting the number of edge-cloud servers needed in an edge-cloud system. Here, we investigate two most important factors: the system workload, i.e., the number of jobs released by the mobile devices, and the upload and download delay to the remote clouds.

In this experiment, we set all the mobile devices to be located closely to one of the edge-cloud servers, so that the importance of the dispatching policy can be better measured. Fig. 4 shows the average WRT with different numbers of edge-cloud servers under different workloads and delays to the



(a) Different system loads

(b) Different upload/download delay to the remote clouds

Fig. 4. The performance over different number of edge-cloud servers

remote clouds. Since *Nearest* performs very badly in this case, we do not show it due to the scaling of the figures. Obviously, with more edge servers, the algorithms will have smaller total WRT, as shown in Fig. 4(a). However, with 1000 and 3000 jobs, the performance gain becomes very small when adding more edge-servers after the server number reaches 20. In the case of 5000 jobs, the performance gain is still significant when increasing the edge-cloud server number from 20 to 30. Thus, we can see that the system load can heavily affect the number of edge-cloud servers required.

Fig. 4(b) shows the performance with different upload/download delays of the remote cloud servers, which are set as a multiple (e.g., 10, 100 and 500 times) of the average upload/download delay of the edge-cloud servers. With more edge-cloud servers, the average WRT is reduced due to the decrease of the jobs dispatched to the remote clouds that suffer from large delays. We can see that the performance gain from adding more edge-servers is greater when the remote delay is larger. For example, we should preferably have nearly 50 edge-cloud servers in the 500x case, and only about 20 servers in the 10x case.

VI. Conclusion

In this paper, we study the online job dispatching and scheduling problem in edge-cloud systems where jobs are released in arbitrary order and times by mobile devices and offloaded to unrelated servers with both upload and download delays. We pose a general model for this problem with the objective to minimize the total weighted response time of all the jobs. We propose the first online approximate algorithm, called `OnDisc`, and prove that it is $(1+\varepsilon)$-speed $O(1/\varepsilon)$-competitive for any constant $\varepsilon \in (0,1)$ based on the speed augmentation model. We also show that `OnDisc` can be easily implemented in distributed systems. Our simulations based on real-world workload traces show that our algorithm can run efficiently and dramatically reduce the total weighted response time in edge-clouds compared with heuristic algorithms. Here, our proposed algorithm is clairvoyant, which means we know the processing time of a job at a server before its computation is completed. An interesting extension of this work is to investigate the non-clairvoyant model.

Acknowledgment

This work is supported in part by NSFC Grant 61502201, China National Funds for Distinguished Young Scientists No. 61625205, Key Research Program of Frontier Sciences of CAS, No. QYZDY-SSW-JSC002, NSFC Grant 61520106007, NSF ECCS-1247944, NSF CMMI 1436786, NSF CNS 1526638, Hong Kong RGC CRF Grant C7036-15G, and NSF-Guangdong Grant 2014A030310172.

References

[1] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *EuroSys 2011*.
[2] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *INFOCOM 2012*.
[3] Z. Han, H. Tan, G. Chen, R. Wang, Y. Chen, and F. C. M. Lau, "Dynamic virtual machine management via approximate markov decision process," in *INFOCOM 2016*.
[4] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 1991–1995, 2012.
[5] M. Jia *et al.*, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," in *INFOCOM 2016*.
[6] Z. Xu *et al.*, "Efficient algorithms for capacitated cloudlet placements," *IEEE Trans. on Parallel and Distributed Systems*, vol. 27, no. 10, 2016.
[7] ——, "Capacitated cloudlet placements in wireless metropolitan area networks," in *Local Computer Networks (LCN) 2012*.
[8] R. Urgaonkar *et al.*, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205–228, 2015.
[9] Y. Zhang, D. Niyato, and P. Wang, "Offloading in mobile cloudlet systems with intermittent connectivity," *IEEE Transactions on Mobile Computing*, vol. 14, no. 12, pp. 2516–2529, 2015.
[10] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *INFOCOM 2016*.
[11] Y. Li, S. H.-C. Jiang, H. Tan, C. Zhang, G. Chen, J. Zhou, and F. Lau, "Efficient online coflow routing and scheduling," in *MOBIHOC 2016*.
[12] M. M. Amble *et al.*, "Content-aware caching and traffic management in content distribution networks," in *INFOCOM 2011*.
[13] S. Wang *et al.*, "Dynamic service migration in mobile edge-clouds," in *IFIP Networking 2015*.
[14] N. Garg and A. Kumar, "Minimizing average flow-time: Upper and lower bounds," in *FOCS 2007*.
[15] 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 *STOC 2009*.
[16] M. S. Gordon *et al.*, "Comet: code offload by migrating execution transparently," in *OSDI 2012*.
[17] M. Satyanarayanan *et al.*, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.
[18] I.-H. Hou, T. Zhao, S. Wang, and K. Chan, "Asymptotically optimal algorithm for online reconfiguration of edge-clouds," in *MOBIHOC 2016*.
[19] S. Im and B. Moseley, "An online scalable algorithm for minimizing lk-norms of weighted flow time on unrelated machines," in *SODA 2011*.
[20] S. Anand, N. Garg, and A. Kumar, "Resource augmentation for weighted flow-time explained by dual fitting," in *SODA 2012*.
[21] L. Schrage, "Letter to the editor – a proof of the optimality of the shortest remaining processing time discipline," *Operations Research*, vol. 16, no. 3, pp. 687–690, 1968.
[22] L. Tong and W. Gao, "Application-aware traffic scheduling for workload offloading in mobile clouds," in *INFOCOM 2012*.
[23] F. Liu *et al.*, "Appatp: An energy conserving adaptive mobile-cloud transmission protocol," *IEEE Transactions on Computers*, vol. 64, no. 11, 2015.
[24] X. Chen *et al.*, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. on Networking*, Sept, 2015.
[25] C. Reiss *et al.*, "Google cluster-usage traces," Technical Report.
[26] A. Chandra, J. Weissman, and B. Heintz, "Decentralized edge clouds," *IEEE Internet Computing*, vol. 17, no. 5, pp. 70–73, 2013.
[27] L. Tawalbeh *et al.*, "Large scale cloudlets deployment for efficient mobile cloud computing," *Journal of Networks*, vol. 10, no. 01, 2015.
[28] X. Ma *et al.*, "Game-theoretic analysis of computation offloading for cloudlet-based mobile cloud computing," in *MSWiM 2015*.