Task Scheduling with Precedence and Placement Constraints for Resource Utilization Improvement in Multi-User MEC Environment

Bowen Liu[a,b], Xiaolong Xu[a], Lianyong Qi[c], Qiang Ni[d], Wanchun Dou[a,b,*]

[a]*State Key Laboratory for Novel Software Technology, Nanjing University, China*
[b]*Department of Computer Science and Technology, Nanjing University, China*
[c]*School of Information Science and Engineering, Qufu Normal University, China*
[d]*School of Computing and Communications, InfoLab21, Lancaster University, U.K*

Abstract

Efficient task scheduling improves offloading performance in mobile edge computing (MEC) environment. The jobs offloaded by different users would have different dependent tasks with diverse resource demands at different times. Meanwhile, due to the heterogeneity of edge servers configurations in MEC, offloaded jobs may frequently have *placement constraints*, restricting them to run on a particular class of edge servers meeting specific software running settings. This spatio-temporal information gives the opportunity to improve the resource utilization of the computing system. In this paper, we study the scheduling method for the jobs consisting of dependent tasks offloaded by different users in MEC. A new task offloading scheduler, Horae, is proposed to not only improve the resource utilization of MEC environment but also guarantees to select the edge server which could satisfy placement constraints for each offloaded task. Concretely, considering the fact that each job would experience slack time as a result of competing for limited resource with other jobs in MEC, Horae minimizes the sum of all slack time values of all the jobs while guaranteeing placement constraints, and therefore improve the resource utilization of the system. Horae was validated for its feasibility and efficiency by means of extensive experiments, which are presented in this

[*]Co-corresponding authors.

Email addresses: `liubw@smail.nju.edu.cn` (Bowen Liu), `njuxlxu@gmail.com` (Xiaolong Xu), `lianyongqi@gmail.com` (Lianyong Qi), `q.ni@lancaster.ac.uk` (Qiang Ni), `douwc@nju.edu.cn` (Wanchun Dou)

paper.

Keywords:
Mobile Edge Computing, Offloading, Precedence Constraints, Resource Utilization

1. Introduction

Currently, pervasive mobile computing and the Internet of Things are driving the rapid development of many new compute-demanding and latency-sensitive applications[1], such as virtual reality (VR), augmented reality (AR) and social gaming [2][3]. Due to the limitation of computation resources and storage capacity in mobile devices, mobile cloud computing (MCC) has been introduced enabling mobile devices to offload their tasks to a more powerful and resourceful cloud [4]. Nevertheless, offloading to the remote cloud can induce significant delays, especially for the scenario that a large amount of data needs to be transferred between the mobile device and the cloud. Besides, massive data going through a centralized network can cause network congestion [5]. Hence, Mobile Edge Computing (MEC) paradigm was introduced to deploy computing functionalities (a.k.a mobile edge servers) at the network edge, thereby meeting the latency requirements of latency-sensitive mobile applications and saving backhaul network bandwidth [6].

Mobile applications and mobile devices have been developing rapidly in recent years[7][8]. Application offloading has been a hot topic owing to the appearance of cloud computing, MEC and increasing computational pressure on mobile devices. Each application can be offloaded in coarse-grained application level [9][10], etc. Application-level offloading means that the whole mobile application functions are executed in the mobile edge server side (e.g., by virtual machines (VMs)), and each User Equipment (UE) runs like a thin client. Actually, application-level offloading enjoys simpler operation and lower programming difficulty but it induces a larger computation workload on edge nodes including the time consumed for application initialization in the VMs. For the task-level offloading, it could give the opportunity to release the intensive workload on mobile devices by distributing its computational tasks to run simultaneously on different edge servers, and therefore improves the performance although needing a more sophisticated task scheduler over the system. In this paper, we only focus on the design of task-level offloading method.

It should be admitted that diverse coexisting offloaded tasks from many users contend for the limited network bandwidth and computation resource among mobile edge servers in a shared MEC environment. MEC operators usually aim at maximizing the throughput of the whole computing system. Besides, to deliver fast and flexible resource provisioning, effective resource management and allocation method are also indispensable for the MEC environment [11][12]. Consequently, an ideal task scheduler is needed to make task scheduling decisions that the overall offloading leads to reduced completion time, and therefore attain high resource utilization over the whole computing system.

In addition, due to the heterogeneity of edge servers configurations and resource limitation in MEC [13], offloaded jobs frequently may have *placement constraints*, restricting them to run on a particular class of edge servers meeting specific software running settings. Also, it should not be ignored the fact that there are *precedence constraints* among tasks in the context of multi-stage application jobs [14]. It is very common in the real world that a task i can be dependent on another task j in the same application if the consumer computation stage of task i is the producer of j. In other words, *precedence constraints* between tasks impose a partial order. Specifically, if task i precedes task j, execution of task j cannot start until task i is finished.

As a result, in the presence of placement constraints and precedence constraints between tasks, scheduling tasks from different jobs in MEC involves some non-trivial technical challenges. Hence, in this work, we consider interdependent tasks scheduling with possible data communication between them as well as placement constraints in an MEC environment. Each task may have predecessor tasks that must be completed before the task can start.

The contributions of this work are as follows:

- For mobile edge computing environment with multiple users, we define a new resource utilization metric called *slack time* of the jobs, and we formulate a problem of slack time minimization in scheduling jobs from different users with precedence constraints and placement constraints, over a MEC system with heterogeneous processors and communication delay.

- We observe that minimize the sum of slack time values of all the jobs could lead to high resource utilization in MEC. Therefore, we propose a heuristic algorithm, termed *Horae*, which utilizes the binary-relaxed solution to make task placement decisions for every offloaded job from different users while minimizing the sum of all slack time values of all the jobs.

- Through randomly generated task trees for different jobs, we study the impact of the number of mobile users on the performance of *Horae*. We further compare Horae with the greedy heuristic algorithm (i.e., ITAGS [14]) to demonstrate the superior effectiveness of our solution in terms of resource utilization.

The reminder of this paper is organized as follows. Section II reviews related works. Section III presents the system model and formulates the problem. Section IV develops the *Horae* method and analyzes its performance. Section V performs simulations, followed by the conclusion in Section VI.

2. Related Work

In this section, we briefly present the research works that are most relevant to our proposed framework. Currently, offloading technique has been a hot topic in both academia and industry. Many works have been proposed to investigate the issues involved in computation offloading in MEC (e.g., energy consumption [15, 14], fairness [16], service caching [13], resource allocation [17], and so on).

2.1. Single User Scenario

Zhang *et al.* [15] considered a Fog cluster with a single mobile device and a number of Fog nodes randomly distributed around the mobile device. The authors divided the Fog nodes into two catagories: *active* fog nodes and *passive* fog nodes according to the sensitivity of energy consumption. They mainly concentrated on the proper decision of the offloading Fog node set N and the offloading load amount for each Fog node in set N. By taking the Fog node's time-energy efficiency and priority into account, a fairness scheduling metric is constructed for each Fog node. The Fair Task Offloading (FTO) scheme selects offloading Fog nodes according to these fairness metrics and then offloads subtasks to the selected nodes based on a rule that minimizes the task delay. However, the work has limitation in the real world in that not all tasks can be divided into a series of subtasks which can be handled in parallel on the nearby Fog nodes. Sundar *et al.* [14] studied the scheduling decision for an application consisting of dependent tasks, and formulated an optimization problem to find the offloading decision that minimizes the overall application execution cost measured by energy consumption, subject to an application completion deadline. This work is similar to our study because it also take precedence constraints of tasks into consideration, but it only considered the task placement decision for offloaded job which can utilize the whole MEC environment in monopolize and ignore the fact that MEC usually accompanied with different jobs from multiple users at any time. Although, the proposed algorithm in [14] can also be utilized by combing each job from different user into one large job, its aim is to minimize the energy consumption for the whole hybird edge/cloud system instead of resource utilization. Also, it still lacks of consideration of *placement constraints* (illustrated in the next subsection) for each job. In a word, these works focus on single user scenario and assume that the the single user could utilize the whole MEC environment exclusively [15, 14], which we think is not practical and would induce low resource utilization.

2.2. Multi-user Scenario

When it comes to multi-user scenario, fairness always exists in resource-limited system, and MEC also can not be an exception. In multi-user MEC environment, the computation resources of the MEC nodes, and communication resources between mobile devices and edge servers (e.g., bandwidth, and power) are shared. Du *et al.* [16] tackled the computation offloading problem in a mixed fog/cloud system by jointly optimizing the offloading decisions and the allocation of computation resources, transmit power, and radio bandwidth while guaranteeing user fairness and maximum tolerable delay. To ensure the fairness of all the UEs, they minimized the maximum cost among all the UEs while meeting the maximum delay constraints. The limitation of this work is that only one Fog node was considered in the analysis.

Xu *et al.* [13] studied joint service caching and task offloading for MEC-enabled dense cellular networks because they noticed that mobile services are heterogeneous in terms of not only required resources (e.g. online Matlab and AR services have different CPU and storage requirements [18]) but also popularity/demand among users. Therefore, the workload of different tasks which offloaded by multiple mobile users may require vastly different amounts of resources (e.g., CPU, memory, and storage).

Meanwhile, driven by the idea of DRF which have been used in multiple type of resource allocation in an abstract single resource pool [19] and Dominant Resource Fairness for Heterogeneous servers (DRFH) which could be used in multiple type of resource allocation among heterogeneous servers [20], Meskar *et al.* [17] considered the problem of fair multi-resource allocation for MEC, and claimed that the shared wireless bandwidth should be considered as an external resource among all the resource for MEC, and designed a multi-resource allocation mechanism that extends the notion of dominant resource fairness to accommodate an external resource.

Fair resource allocation and high resource utilization are beneficial to users and MEC operators respectively. However, it must be admitted that ensuring the optimal fairness of resource allocation and the high utilization rate of resources are often conflicting goals, which cannot be realized simultaneously. There is a tradeoff between these two goals. Due to the limitation of space, this paper only studies from the perspective of resource utilization, and the analysis of tradeoffs will be left for later work.

3. System Modeling and Problem Formulation

In this section, we present our models and motivate the need to improve resource utilization for offloading dependent tasks in a multi-user MEC environment. To simplify the discussion, the mathematical notations are summarized in Table 1.

3.1. Service Region and Processors

We consider a mobile network of numerous mobile edge servers, and each mobile edge server is endowed with computing resources and hence can provide offloading service to end users in its wireless range. The whole network is divided into M disjointed small regions, indexed by $\mathcal{M}$. Mobile user in each region r can reach a set of mobile edge servers in the wireless range, denoted by F_r, and the size of F_r is denoted as $|F_r|$. We also assume that there are some local processors installed in these mobile edge servers, and these processors may have different processing capabilities due to the heterogeneous MEC environment. In common with [14], we assume that each processor executes one task at a time, while the other tasks offloaded to the processor waiting in a task queue. Suppose that the delay per unit data transfer between processor i and j is denoted as $d_{i,j}$. To enable tractable analysis, we assume $d_{i,j} = d_{j,i}$ and $d_{i,j} = 0$ if $i = j$.

Without loss of generality, each region could have its own offloading scheduler which aims at its predefined scheduling objective (e.g., minimizing average job completion time, minimizing maximum job lateness, minimizing the

Table 1: Mathematical Notations

Notation	**Description**
i, j	A tack unit of a job
J	A complete job consist of several tasks
$\mathcal{M}$	Number of disjointed network regions in the whole network
F_r	Mobile edge server set of region r
$\mathcal{P}_r$	Total number of processors in region r
$d_{i,j}$	Delay per unit data transfer between processor i and j
v_i	Processing speed of processor i
$e^J_{i,j}$	Required data transfer between task i to j in job J
$VP(i)$	Immediate predecessors of task i
$VS(j)$	Immediate successors of task j
$t^J_{i,p}$	Execution time of task i of job J on processor p
G_J	Directed acyclic graph G of job J
$\mathcal{V}_J$	Task set of job J
$\mathcal{E}_J$	Precedence constraint set of job J
T^i_J	Task of i in job J
$slack^J$	Slack time of Job J
DS	Dummy node to trigger the process of job
DE	Dummy node to indicate the finish of job
C^J_{share}	Completion time of job J when scheduled in a shared system
$C^J_{monopolize}$	Completion time of job J when it could use the resource monopolistically
$x^i_{J,p,l}$	Scheduling decision for task i of job J on processor p in index l
$c^J_{i,k}$	Variable to indicate whether a given processor k can satisfy placement constraint of task i in job J
N_r	Total number of tasks in region r
N^r_{jobs}	Number of jobs in region r
N^J_{tasks}	Number of tasks of job J
F^i_J	Finish time of task i of job J

number of tardy jobs, and so on). A scheduler could keep its high availability by leveraging some fault-tolerant coordination protocols (e.g., zookeeper [21]) which could perform the discovery of the mobile edge servers in the region, health checking, implementation of election protocol, and communication between mobile edge servers.

3.2. Precedence Constraints

In a shared mobile edge computing region, diverse coexisting jobs contend for the limited computation and network resource. Not until all dependent tasks have completed will the associated job complete. In this paper, we bring *Precedence Constraints* to describe this phenomenon. Consider a computation job J which could be partitioned into different dependent tasks, and the dependency is modeled as a directed acyclic graph (DAG) $G_J = \langle \mathcal{V}_J, \mathcal{E}_J \rangle$ where $\mathcal{V}_J$ is the set of partitioned tasks and $\mathcal{E}_J$ is the set of corresponding edges. The edge (i, j) in $\mathcal{E}_J$ specifies that there is some required data transfer, $e^J_{i,j}$, from task i to task j and hence, task j cannot start before the transmission finishes. The set $VP(i)(VS(i))$ of all immediate predecessors (successors) of job J is given by

$$VP(j) = \{i|(i,j) \in \mathcal{E}_J\} \quad (VS(i) = \{j|(i,j) \in \mathcal{E}_J\}) \tag{1}$$

In fact, that scheduling multiple jobs simultaneously is equal to scheduling multiple DAGs at the same time. If task i and task j are scheduled on different processors q and r respectively, it is obvious to notice that the communication delay is $e^J_{i,j}d_{q,r}$. If task i is executed on processor p exclusively (which means that if task i arrive at the processor, it could be executed immediately), the execution time is $t^{J*}_{i,p}$. In practice, because the fact that there would several tasks which offloaded by other users pending for execution, the actual completion time for task i executed on processor p is denoted as $t^J_{i,p}$ which is usually larger than $t^{J*}_{i,p}$. In fact, the execution time and data transfer requirement could be obtained by applying a program profiler such as MAUI [22].

There are two obvious solutions to schedule multiple DAGs [23].

1. Schedule the DAGs one after the other using any single-DAG scheduling algorithm.

2. All DAGs could be combined into a bigger DAGs by introducing same immediate predecessor and successor, and then apply any single-DAG scheduling algorithm on the combined DAG.

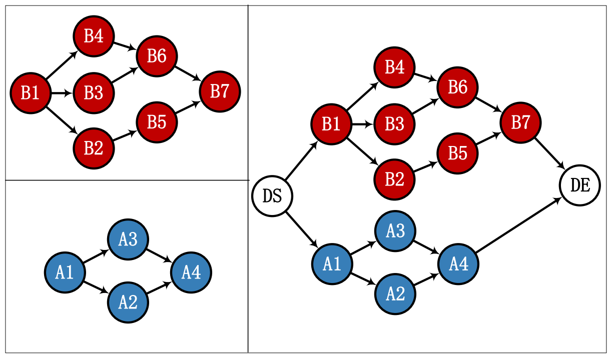


Figure 1: Example of generation process of combined DAG.

However, the disadvantage of the first scheduling solution is that the resource utilization of the system will be in a low state for a long time. Besides, with the increase in the number of DAGs, this scheduling idea will further extend the response time of the tail-end jobs. Although the second scheduling idea could improve the resource utilization of the entire computing system, single-DAG scheduling algorithm could not provide scheduling optimization for each sub-DAG. In extreme cases, there are still cases where some jobs are delayed in scheduling. Following the terminology in the field of operating system research, the job that is delayed in scheduling is referred to the "starving" job.

In this paper, we focus on the second solution, and try to explore how to make use of resources in the actual scheduling process while ensuring that there is no "starving" job during the scheduling process.

Figure 1 illustrates the generation process of combined DAG from two DAGs. Specifically, each DAG could have another two logical dummy nodes (i.e., tasks having zero execution time and varied communication cost). One dummy node (i.e., DS) is inserted at the start to each DAG and the other one (i.e., DE) is inserted at the very end location of each DAG. These two dummy nodes indicate the communication cost between mobile devices and mobile edge servers. Hence, both DAGs could have the same predecessor and successor logically, but have different communication cost at the start stage and finish stage of offloading. Again, note that Figure 1 does not show the communication cost between each pair of dependent tasks.

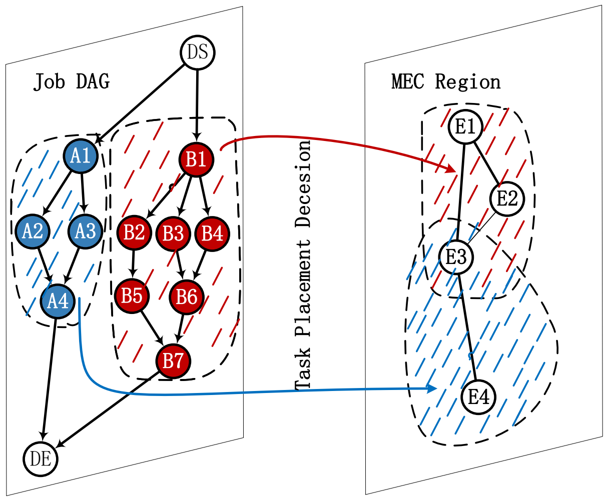


Figure 2: An instance of task placement decisions.

3.3. Placement Constraints

Over time, a MEC environment typically includes heterogeneous mobile edge servers with different configurations. Incompatibilities between mobile edge server configurations and prerequisites of task execution are often encountered. Unlike the cloud which has huge diverse resources, the limited computing and storage resources of mobile edge server allow only a small set of services to be cached at the same time [13]. Therefore, not all mobile edge servers could be utilized for job offloading if the selected mobile edge server has no corresponding suitable service library. To this end, we bring *placement constraints* to describe this phenomenon. Placement constraints must be satisfied appropriately (e.g., the requirement of Java Runtime Environment) if used for a specific task offloading. The offloading scheduler is required to make task placement decisions for every task in the job DAG with placement constraints.

As an example, Figure 2 demonstrates the task placement decisions. Concretely, the left-hand side shows an example of a combined job DAG plane, which includes two jobs offloaded by different users. The right-hand side plane indicates the MEC region which includes four heterogeneous configurations. Due to the fact that different jobs may have different placement constraints, Figure 2 assumes that only mobile edge server nodes 1, 2, and 3 can satisfy the placement constraints of job B, mobile edge server nodes 3 and 4 can satisfy the placement constraints of job A, respectively. Note that mobile edge server node 3 can not only execute tasks of job A but also job B. The soft constraints of job A and B are optional and implicit and soft constraints are optional to satisfied and depend on situations (e.g., if tasks with precedence constraints execute on the same node, then it will reduce the communication cost between tasks but may induce extra task waiting time). We use the notation T_i^J to denote the task i of job J, and we will omit superscript J when it causes no ambiguity in the rest of the paper. As shown in equation (2), $c_{i,k}^J$ can indicate whether task T_i^J can be placed on the mobile edge server k. Note that this equation is also equivalent to whether the mobile edge server k can satisfy the placement constraint of task i in job J.

$$c_{i,k}^J = \begin{cases} 1, & \text{Processor } k \text{ can satisfy } \textit{placement} \\ & \textit{constraint} \text{ of task } i \text{ in job } J \\ 0, & \text{Otherwise,} \end{cases} \tag{2}$$

3.4. High Resource Utilization Objective

In practice, it could have multiple mobile users in each edge service region at any time. Hence, coexisting offloaded tasks from different users contend for both computing resource and communication bandwidth in a shared edge computing environment. In common to cloud computing which provides a pool of resources for the users to run their applications according to "pay per use" fashion, the improvement of resource utilization (CPU and network resource in our case), along with the increased rate of processed jobs, implies a higher profit rate for the public MEC provider. To improve resource utilization in MEC, operators need to finish as many jobs as possible, each in the fastest possible way. In the context of scheduling, each job would experience delays as a result of sharing the resources in MEC environment with other jobs. We define *slack time* to measure the delay of each job. The amount of slack time of job i is the difference between its possible completion time under a shared MEC environment and its earliest possible completion time under a monopolistic MEC environment. An ideal scheuduler keeps resources busy (as in load balancing), and it allows multiple users to share whole resources effectively. The objective of this work is to provide an isolation guarantee between contending offloading jobs while minimizing the average job completion time (JCT). The completion time and the slack time are indispensable. For mobile edge computing environment with multiple users, if the constraints are met slack time is part of the completion time. Because the running time is fixed and there are constraints on the order of tasks, if the total of the slack time is the minimum so that the completion time will be the minimum.

To ensure performance isolation between users, a task scheduler should provide a *minimum progress guarantee* for each offloaded job.Therefore, the slack time value of the job i in a shared region is defined as

$$slack^i = (C^i_{share} - C^i_{monopolize})^2, \tag{3}$$

where $C^i_{monopolize}$ is the completion time of the job when assume it could have all the available resource in the region, and C^i_{share} is the completion time of the job when scheduled along with all the other jobs in the region. Intuitively, $C^i_{monopolize}$ could be considered as the optimal completion time of job i, and $C^i_{share} \geq C^i_{monoplize}$ always holds, and the lower the slack time value means the closer to the optimal completion time. Meanwhile, it is intuitive to notice that the lower the slack time value means the higher resource utilization among the system.

3.5. List Scheduling

Considering the fact that computing service requests from different users arrive at different times, the actual scheduler will continuously receive service requests of new jobs. At any time, the scheduler divides all the jobs into two categories: jobs to be completed and jobs marked as completed. Whenever a new job comes, the scheduler needs to save the scheduled context to take over the newly arrived job. After recording the basic information of the job (placement constraints and precedence constraints between tasks, etc.), mark it as a job to be completed, and then take all the jobs marked as completed into consideration to make scheduling decisions.

Considering the following on-line *list scheduling* algorithm: whenever a mobile edge server is idle, choose any available task to start processing on that mobile edge server, where a task j is available, if all of its predecessors ($VP(j)$) have finished processing; if none are available, wait until the next available task, and continue scheduling then. Again, once a processor P_i begins to execute a task T^j_J, it works without interruption on T^j_J until completion of that task, taking altogether p^j_J/v_i units of time. If given a liner processing ordering L of all tasks in job J. In general, at any time t a processor P_i completes a task, it immediately and instantaneously scans the list L (starting from the beginning) until it comes to the *first task* T_J^j which has not yet begun to be executed. if all the predecessors of T_J^j have been completed by time t then P_i begins working on task T_J^j. Otherwise P_i proceeds to the next task in L which has not yet begun to be executed, etc. If P_i proceeds through the entire list L without finding a task to execute then P_i becomes idle. P_i remains idle until some other P_j completes a task at which time P_i (and of course P_j) immediately scans the list L as before for possible tasks to execute. If these two processors simultaneously attempt to begin the same task, the target scheduler should make a specific decision to allocate the task. Different scheduling decisions can induce different performance in terms to the same job set. The algorithm of list scheduling could be illustrated in Algorithm 1.

Algorithm 1: List scheduling algorithm

```
1 foreach Unfinished task i do
2     if all predecessor have finished then
3         execute task without interruption
4     end
5     else
6         idle and wait next checkpoint
7     end
8 end
9 return result;
```

3.6. Case Study

To motivate the need for our proposed scheduling method (*Horae*), a case study is investigated in this section, and we give the definition of resource utilization of MEC environment as follows.

***Definition* 1** (Resource Utilization of MEC Environment)**.** *Given a time period* $[t, t + \Delta]$ *and a limited collection of heterogeneous mobile edge servers* $Es_r = \{es_1, es_2, \ldots, es_r\}$ *with resource type* $R = \{r_1, r_2, \ldots r_Q\}$ *where* Q *is the number of resource types. The resource utilization of resource type* q *is denoted as* $U_q = \frac{\int_t^{t+\Delta} \sum_i^r c_q^i \, dt}{\Delta \sum_i^r c_q^{i,max}}$ where c_q^i and $c_q^{i,max}$ are the instantaneous and maximum processing capability of resource q on edge server es_i, respectively.

Table 2: Processing Time and Communication Delay

Tasks	Processing Time (ms)	Communication Cost [$\star$](ms)
DS	0	$\overset{1}{\hookrightarrow}$A1 $\overset{2}{\hookrightarrow}$B1
A1	1	$\overset{1}{\hookrightarrow}$A2 $\overset{1}{\hookrightarrow}$A3
A2	2	$\overset{1}{\hookrightarrow}$A4
A3	2	$\overset{1}{\hookrightarrow}$A4
A4	2	$\overset{1}{\hookrightarrow}$DE
B1	1	$\overset{1}{\hookrightarrow}$B2 $\overset{1}{\hookrightarrow}$B3 $\overset{1}{\hookrightarrow}$B4
B2	3	$\overset{1}{\hookrightarrow}$B5
B3	5	$\overset{1}{\hookrightarrow}$B5
B4	3	$\overset{1}{\hookrightarrow}$B6
B5	4	$\overset{1}{\hookrightarrow}$B7
B6	2	$\overset{1}{\hookrightarrow}$B7
B7	2	$\overset{1}{\hookrightarrow}$DE

[$\star$] $\overset{x}{\hookrightarrow}$y means that x ms delay would be took for the task outputs its processed result to task y.

To simplify the analysis and discussions, we assume that each mobile edge server is identical in terms of CPU processing speed, and the communication cost of any pair of dependent tasks that run on two different processors is a constant in the following case study. Also, we assume each task will receive all the processing capability from each processor and all bandwidth of an embedded network interface. Considering a scheduling requirement for 2 jobs offloaded from different users, and each job involves different dependent tasks which have been depicted in Figure 1. Table 2 shows the processing times of all tasks and communication cost of each pair of dependent tasks. For example, the processing time of A1 is 1 ms, and it will take 1 ms communication cost for the initiation of task A2.

If the two jobs are scheduled one after the other, a possible scheduling scheme could be described in Figure 3(a). If we schedule job B firstly and then move to job A, job B and job A will finish at 15 ms and 25 ms, respectively. Note that mobile edge server E3 and E4 were idle before time 5 ms in this scheduling scheme.

Obviously, this induces low resource utilization of the whole computing system and long response time for the job A. If we schedule job A and B simultaneously, job A could have progressed before time 5 ms, and Figure 3(b) shows an example scheduling scheme for these two jobs. Job B and job A will finish at 15 ms and 10 ms in this scheduling scheme, respectively. Although the completion time of job B is still 15 ms, the completion time of job A is advanced by 15 ms (the completion time is advanced from 25 ms to 10 ms). Moreover, it is not hard to notice that the resource utilization is higher than the scheduling scheme in Figure 3(a).

3.7. Problem Formulation

One may actually consider the scheduling of parallel processors as a two-step process. First, one has to determine which tasks have to be allocated to which processors; second, one has to determine the sequence of the tasks allocated to each processor. The scheduling decision for each task in each region could be described as follows:

$$x^i_{j,p,l} = \begin{cases} 1 & \text{if task } j \text{ of job } i \text{ on processor } p \text{ in index } l, \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

for all $i = 1, 2, \ldots, N^r_{\text{jobs}}, \quad j = 1, 2, \ldots, N^i_{\text{tasks}}, \quad p = 1, 2, \ldots, \mathcal{P}_r$ and $l = 1, 2, \ldots N_r$, where $N^r_{\text{jobs}}, N^i_{\text{tasks}}$ and N_r are the number of jobs in region r, the

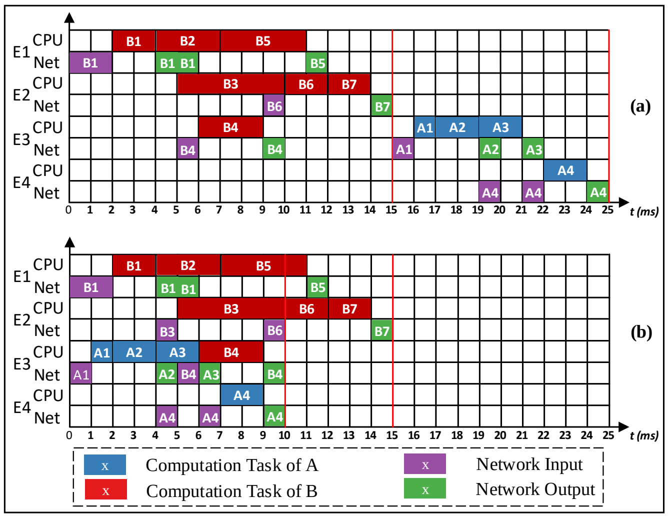


Figure 3: An Example Scheduling Scheme for Job A and B

number of tasks of corresponding job i and the total number of tasks in the region, respectively. Considering the fact that each task should be scheduled to only one of the existing queue index in the processors, and we assume each processor own a virtual task queue whose maximum length is equal to N_r in case of all tasks are scheduled to one processor, and N_r could be calculated as $\sum_{i=1}^{N^r_{\text{jobs}}} N^i_{\text{tasks}}$ obviously. Therefore, we have the following constraint in each MEC region $r \in \mathcal{M}$:

$$\sum_{p=1}^{\mathcal{P}_r} \sum_{l=1}^{N_r} x^i_{j,p,l} = 1, \ \forall i = 1, \dots, N^r_{\text{jobs}}, \ j = 1, \dots, N^i_{\text{tasks}}. \tag{5}$$

Furthermore, each index on each processor can be assigned to at most one task, which is given by

$$\sum_{i=1}^{N^r_{\text{jobs}}} \sum_{j=1}^{N^i_{\text{tasks}}} x^i_{j,p,l} \le 1, \quad \forall p = 1, \dots, \mathcal{P}_r, \ l = 1, \dots, N_r. \tag{6}$$

Also, the scheduling decision should follow the placement constraints, which could be described as follows:

$$\sum_{l=1}^{N_r} x^i_{j,p,l} \le c^i_{j,p}, \tag{7}$$
$$\forall i = 1, \dots, N^r_{\text{jobs}}, \ j = 1, \dots, N^i_{\text{tasks}}, \ p = 1, \dots, \mathcal{P}_r,$$

where $c^i_{j,p}$ is a constant value if given a specific job description and MEC environment.

Suppose the finish time of task j of job i is denoted as F^i_j, and it is scheduled on index l' in processor v' (which means that $x^i_{j,v',l'}$ is set to 1). On the one hand, according to the task precedence constraints in each job, we have

$$\begin{aligned} F^i_j \geq \max_k & \left(F^i_k + \sum_{v=1}^{\mathcal{P}_r} \sum_{v'=1}^{\mathcal{P}_r} \sum_{l=1}^{N_r} \sum_{l'=1}^{N_r} e^i_{k,j} d_{v,v'} x^i_{k,v,l} x^i_{j,v',l'} \right) \\ & + \sum_{v'=1}^{\mathcal{P}_r} \sum_{l'=1}^{N_r} x^i_{j,v',l'} t^i_{j,v'}, \\ & \forall j = 1, \ldots, N^i_{\text{tasks}},\ (k,j) \in \mathcal{E}_i \end{aligned} \tag{8}$$

if task k is one of the immediate dependent tasks of task j in job i, and it is scheduled on index l in processor v following the task placement decision $x^i_{k,v,l}$. Notice that the inequality holds when task j should wait the all tasks which placed before index of l in processor v finish although all predecessors of task j have finished.

On the other hand, we have

$$F^i_j = \sum_{i'=0}^{N^r_{\text{jobs}}} \sum_{j'=0}^{N^{i'}_{\text{tasks}}} \sum_{l=0}^{l'} x^{i'}_{j',v',l} t^{i'}_{j',v'}, \tag{9}$$

which means that the finish time of task j of job i not only depends on the placement decision ($x^i_{j,v',l'}$), but also satisfies constraint (8).

The relationship between task finish time F^i_j and C^i_{share} could be described as follows:

$$C^i_{share} = \max_j F^i_j, \quad \forall i = 1, \ldots, N^r_{\text{jobs}}. \tag{10}$$

We assume that different tasks on one specific processor should be processed sequentially. The following constraint ensures that until one index on a processor is occupied, tasks cannot be assigned to subsequent indexes:

$$\begin{aligned} \sum_{i=1}^{N^r_{\text{jobs}}} \sum_{j=1}^{N^i_{\text{tasks}}} x^i_{j,p,l} - \sum_{i=1}^{N^r_{\text{jobs}}} \sum_{j=1}^{N^i_{\text{tasks}}} x^i_{j,p,(l-1)} \leq 0, \\ \forall p = 1, \ldots, \mathcal{P}_r,\ l = 2, \ldots, N_r. \end{aligned} \tag{11}$$

The new problem is to identify the schedule that minimizes the total sum of slack time values of all the jobs in a service region. This can be formulated as an optimization problem as follows:

$$
\begin{aligned}
& \underset{\{x^i_{j,p,l}\}}{min} \sum_{i=1}^{N^r_{\text{jobs}}} slack^i \\
= & \underset{\{x^i_{j,p,l}\}}{min} \sum_{i=1}^{N^r_{\text{jobs}}} (C^i_{share} - C^i_{monopolize})^2 \\
= & \underset{\{x^i_{j,p,l}\}}{min} \sum_{i=1}^{N^r_{\text{jobs}}} (\underset{j}{max}\, F^i_j, -C^i_{monopolize})^2 \\
& s.t. \quad (5)(6)(7)(8)(9)(11) \\
& \qquad x^i_{j,p,l} \in \{0,1\}, \\
& \qquad \forall i = 1, \ldots, N^r_{\text{jobs}},\ j = 1, \ldots, N^i_{\text{tasks}}, \\
& \qquad p = 1, \ldots, \mathcal{P}_r,\ l = 1, \ldots, N_r,\ (k,j) \in \mathcal{E}_i
\end{aligned}
\tag{12}
$$

4. Horae

The optimization problem (12) is a mixed integer program, and it is non-convex due to its non-convex constraints. This problem is NP-hard since it contains the Generalized Assignment Problem (GAP) as a special case, and GAP is NP-hard [24, 25]. Hence, we do not expect to find an optimal solution in polynomial time in this study. Again, $C^i_{monopolize}$ could be considered as a constant value because it is not difficult to notice that to obtain the $C^i_{monopolize}$ for a given job i DAG, mobile edge computing environment settings and task placement constraints is a subproblem of the problem (12) which can be illustrated as follows:

$$\begin{aligned}
&C^i_{monopolize}\left(=\min_{\{x^i_{j,p,l}\}}\max_j F^i_j\right)\\
&\geq \min_{\{x^i_{j,p,l}\}}\max_j\left[\sum_{v'=1}^{\mathcal{P}_r}\sum_{l'=1}^{N_r}x^i_{j,v',l'}t^i_{j,v'}+\right.\\
&\left.\max_k\left(F^i_k+\sum_{v=1}^{\mathcal{P}_r}\sum_{v'=1}^{\mathcal{P}_r}\sum_{l=1}^{N_r}\sum_{l'=1}^{N_r}e^i_{k,j}d_{v,v'}x^i_{k,v,l}x^i_{j,v',l'}\right)\right],\\
&s.t.\quad (5)(6)(7)(8)(9)(11)\\
&x^i_{j,p,l}\in\{0,1\},\\
&\forall j=1,\dots,N^i_{\text{tasks}},\ p=1,\dots,\mathcal{P}_r,\ k\in VP(j),
\end{aligned}\tag{13}$$

Note that this inequality holds equal if there is no additional waiting cost for dependent tasks in job i.

Let

$$C^i_{execution}=\min_{\{x^i_{j,p,l}\}}\max_j\left(\max_k F^i_k+\sum_{v'=1}^{\mathcal{P}_r}\sum_{l'=1}^{N_r}x^i_{j,v',l'}t^i_{j,v'}\right),$$

where $C^i_{execution}$ means the actual execution time cost on processors in MEC region. It is not difficult to note that $C^i_{monopolize}\geq C^i_{execution}$ always holds.

Therefore, we can relax the optimize objective of problem (13) and rewrite the corresponding constraints as

$$\begin{aligned}
C^{i(\mathbf{relaxed})}_{monopolize}=&\min_{\{x^i_{j,p,l}\}}\max_j\left(\max_k F^i_k+\sum_{v'=1}^{\mathcal{P}_r}\sum_{l'=1}^{N_r}x^i_{j,v',l'}t^i_{j,v'}\right)\\
&s.t.\quad (5)(6)(7)(8)(9)(11)\\
&x^i_{j,p,l}\in\{0,1\},\forall p=1,\dots,\mathcal{P}_r,
\end{aligned}\tag{14}$$

by reducing the data communication time cost between arbitrary tasks with precedence constraints executing on different processors. Besides, this relaxed problem could be considered as the classical problem which schedules n tasks on a set of m machines at different speeds. Let C_j denote the completion time of task j. The objective is to find a schedule that minimizes the time needed to finish all of the tasks or, in other words, to minimize $C_{max}=max_j\, C_j$, called the makespan of the schedule. In the scheduling literature where problems are classified in the $\alpha|\beta|\gamma$ notation [26], this problem is denoted as $Q_m|prec|C_{max}$, and it has been addressed by [27, 28] by using $\mathcal{O}(\log\ m)$-approximate algorithms. Note that although this problem does not take the communication time cost between dependent tasks, it could be considered as a upper bound for problem (13) by solving problem (14).

Because the $C^i_{monoplize}$ of each job i is independent in mobile edge region, we could let $C^{i(\mathbf{relaxed})}_{monopolize}$ replace the parameter $C^i_{monopolize}$ in problem (12) to get a approximate solution. Further, note that the original problem (12) is non-convex, and we can consider a binary-relaxted version of it. By replacing each $C^{i(\mathbf{relaxed})}_{monopolize}$ as c^i and using binary relaxation on problem (12), it could be modified as follows:

$$\begin{aligned}\underset{\{x^i_{j,p,l}\}}{min} \sum_{i=1}^{N^r_{\text{jobs}}} \Bigg(\underset{j}{max} \Bigg\{ \underset{k}{max}\, F^i_k + \sum_{v'=1}^{\mathcal{P}_r} \sum_{l'=1}^{N_r} x^i_{j',v',l'} t^i_{j,v'} \\ + \sum_{v=1}^{\mathcal{P}_r} \sum_{v'=1}^{\mathcal{P}_r} e^i_{k,j} d_{v,v'}\, max \Bigg[\Bigg(\sum_{l=1}^{N_r} x^i_{k,v,l} \Bigg) \\ + \Bigg(\sum_{l'=1}^{N_r} x^i_{j,v',l'} \Bigg) - 1, 0 \Bigg] \Bigg\} - c^i \Bigg)^2\end{aligned} \tag{15}$$

Obviously, problem (15) is a convex problem over decision variables $\{x^i_{j,p,l}\}$, and it can be efficiently computed using convex programming solvers such as CVX and CVXPY. Note that replacing constraint $x^i_{j,p,l} = \{0,1\}$ with $x^i_{j,p,l} \geq 0$ is equivalent to allowing a single task to be distributed and executed partially across several processors and indexes. However, the relaxed solution can be leveraged to recover a binary approximate solution to the original problem (12).

4.1. Offline Scheduling

We consider an offline problem of scheduling M jobs $J_1, J_2, \ldots, J_M$ that arrived at time 0. All the problem data (e.g., processing times, placement constraints, precedence constraints for each job, and so on) are known in advance and can be taken into account in the optimization process.

Algorithm 2 shows the pseudocode of the offline scheduling scheme in Horae. Although offline scheduling is unrealistic in the real world, it could be considered as the benchmark of online scheme. Firstly, Horae calculates the optimal finish time for each job with placement constraints in the MEC environment by solving the problem (14). These optimal finish times are considered as the dominant factor for task placement decision of scheduling. Secondly, the ideal fractional solution $\{x^i_{j,p,l}\}^{\textbf{relaxed}}$ is got by solving problem (15). Since this solution allows each task could be divided and placed on different processors, the next stage is to aggregate all the fractional parts of placement decisions to get the highest probability of each task (line 6 - line 12). Lastly, for each processor, horae determine the processing order for tasks according to their original tentative finish time (line 11 - line 20).

As for practical problems $|\mathcal{E}|$ has the same order of magnitude as $|\mathcal{V}|$ [29], Because the offloaded DAGs usually could be represented by sparse graphs (those for which $|\mathcal{E}|$ is much less than $|\mathcal{V}|$), the computational complexity for the Horae-Offline algorithm excluding the time to compute the lower bound solution is $\mathcal{O}(M|\mathcal{P}_r||\mathcal{V}|\text{log}|\mathcal{V}|)$, which is polynomial with respect to the number of the jobs. The time cost for computing the lower bound is dependent on the algorithms used by the solver to arrive the solution. According to [30], the number of iterations for a convex program arriving at the solution is $\mathcal{O}(\sqrt{|\mathcal{V}\mathcal{P}_r|}\text{log}(\frac{|\mathcal{V}\mathcal{P}_r|\mu}{\epsilon}))$, where ϵ and μ are the optimal value and the barrier parameter used in corresponding primal barrier algorithm.

4.2. From Offline to Online

In an online scheduling problem, the MEC environment cannot accurately predict how many jobs from different users will arrive at the platform, and it is impossible to know when these jobs arrive. MEC platform becomes aware of the corresponding information (e.g., dependency constraints, placement constraints and computation) of a job only when the job is released and presented to it. Therefore, the list scheduling algorithm which has been presented in subsection III.E is utilized to cooperate with online version of Horae. In the online version of Horae, each job has its own task processing list. Whenever a mobile edge server becomes idle, it will scan from every task processing list to pick up all the available tasks, and then make a suitable decision to choose which task should be executed immediately on itself.

We focus on self-adjusting strategies for effective task scheduling in locally distributed mobile edge servers that provide offloading services. We assume an online version of Horae architecture consisting of a front-end job repository and a set of heterogeneous, back-end mobile edge servers. The job repository acts as the initial interface between the cluster mobile edge servers and the smart mobile devices, and manage all incoming offloading requests, trying to record and maintain the offloading procedure for all offloaded jobs.

Algorithm 2: Horae (offline version)

```
    Input: Placement Constraints Set
           $\{c^1_{1,1}, c^1_{1,2} \dots c^J_{1,P_r}, c^1_{2,1}, c^1_{2,2} \dots c^1_{2,P_r} \dots c^J_{N^M_{\text{tasks}},P_r}\}$, Jobs DAG set
           $\{\langle \mathcal{V}_1, \mathcal{E}_1 \rangle, \langle \mathcal{V}_2, \mathcal{E}_2 \rangle, \langle \mathcal{V}_3, \mathcal{E}_3 \rangle \dots \langle \mathcal{V}_M, \mathcal{E}_M \rangle\}$
    Output: Placement decisions $\{x^i_{j,p,l}\}$ for all tasks
1   foreach Job $i$ do
2       Get $C^{i(\textbf{relaxed})}_{monopolize}$ by solving problem (14);
3       Set $C^{i(\textbf{relaxed})}_{monopolize}$ as the $c^i$ in problem (15);
4   end
5   Get fractional solution $\{x^i_{j,p,l}\}^{\textbf{relaxed}}$ ;
6   foreach Job $i$ do
7       foreach Task $j$ of job $i$ do
8           $y^i_{j,p} = \sum_{l=1}^{N_r} x^i_{j,p,l}$;
9           $s^i_j = \arg\max_p y^i_{j,p}$;
10          $y^i_{j,p} := \begin{cases} 1, & \text{if } s^i_j = y^i_{j,p}, \\ 0, & \text{otherwise.} \end{cases}$
11      end
12  end
13  foreach Processor $p$ do
14      foreach Job $i$ do
15          foreach Task $j$ of job $i$ do
16              $L_p \leftarrow \emptyset$;
17              if $y^i_{j,p} == 1$ then
18                  $L_p \leftarrow L_p \cup T^i_j$;
19              end
20          end
21          Sort $L_p$ according to $F^i_j$ in a ascending order;
22          Mapping each index of processor $p$ according to the index of
              each task $T^i_j$ in $L_p$ to get $\{x^i_{j,p,l}\}$ ;
23      end
24  end
25  return $\{x^i_{j,p,l}\}$;
```

Considering the fact that any DAG has at least one topological ordering (In general, a typical topological ordering of a DAG is a linear ordering of its vertices such that for every directed edge (u, v) from vertex u to vertex v, u comes before v in the ordering), and many algorithms (e.g., [31] and [32]) have been designed for constructing a topological ordering of any DAG in linear time. In fact, there are similar features in the execution dependencies between tasks within the job. This means that any topological ordering of the DAG corresponds to a method of handling task dependencies within the job. The total number of topological orderings also corresponds to the number of job scheduling methods that can satisfy the task execution dependencies.

Therefore, for each job DAG, we construct a topological ordering to facilitate mobile edge server's processing in the list-scheduling algorithm. Algorithm 3 shows the pseudocode of the task processing list generation for each job. Considering the fact that tasks in most of the jobs could be executed in different mobile edge servers simultaneously, we give priority to the task with the maximum processing time in the processing order list if multiple tasks are available to execute. This heuristic tries to place the shorter tasks more towards the end of the schedule, where they can be used for balancing the workloads.

Due to the fact that Horae aims to find a scheduling scheme that can minimize the total slack time values of all the jobs. When any processor is idle, the processor not only needs to ensure that the task processing is performed according to the task processing list order within each job, but also decides which job processing list should be selected. Whenever a new job is arriving in the Horae-enabled MEC environment, the candidate MEC nodes are generated firstly by checking placement constraints for the job. Then, the job will be added to the unfinished job set to facilitate Horae's scheduling. Some intrinsic job profile for the forthcoming scheduling should be prepared (e.g., arriving time stamp, task processing list, and optimal finish time in the MEC environment). Suppose job J's arriving time is denoted as A^J, and it will finish at F^J if all resources in the current Fog computing environment could be monopolize leveraged (imply that $F^J - A^J = C^{J(\mathbf{relaxed})}_{monopolize}$). When a processor is in idle status on time $T_{schedule}$, then the duration of job J can be calculated by $T_{schedule} - A^J$. The details could be found in Algorithm 4.

In Horae, each mobile edge server greedily selects the task whose corresponding job has the maximum *slack time* value to execute on its processor.

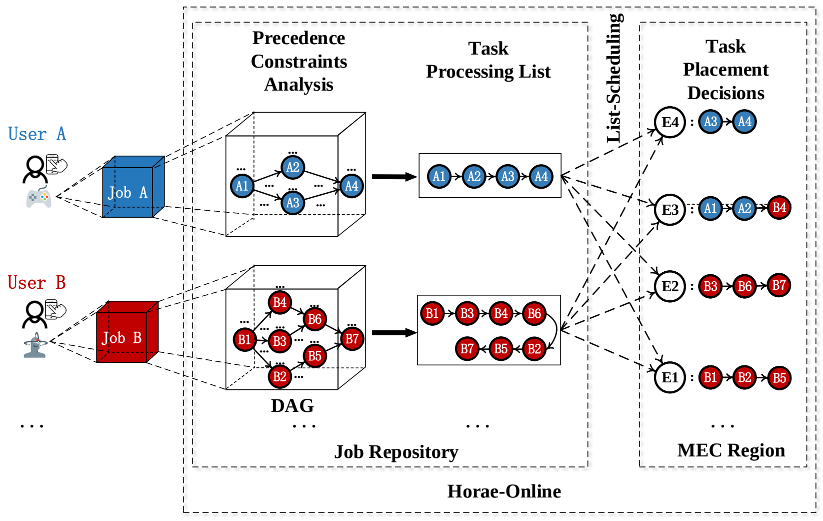


Figure 4: The architecture of Horae-online scheduling system.

We use

$$F^{p}_{est}(h_j, J) = \frac{\sum_{h_j}^{|N^J_{\text{tasks}}|} p^{h_j}_J}{v_p} + T_{current} \tag{16}$$

to measure the estimated finish time of job if scheduled on processor p, where h_j means the index of next unexecuted task of job J in the corresponding topological ordering, $T_{current}$ means the current time. Horae gives priority to the job with the maximal slack time value $(F^{p}_{est}(h_j, J) - F^J)$ to be executed whenever one mobile edge server is available to run the next task. The details of online version of Horae could be seen in Algorithm 5, and the computational complexity for the Horae-Online algorithm is $\mathcal{O}(|\mathcal{V}|^2|\mathcal{P}_r|)$ for each job DAG. Again, for each coming job DAG, the solution of problem (14) is assumed to be known in advance by applying a program profiler such as MAUI [22].

Figure 4 shows the details of the architecture for Horae-Online scheduling system.

5. Performance Evaluation

In this section, we evaluate the performance of our approach by extensive experiments with a comparison to two baseline approaches. All the experiments were conducted on a Windows machine equipped with Intel Core

Algorithm 3: Task processing list generation

```
Input: Job DAG $\langle \mathcal{V}_J, \mathcal{E}_J \rangle$
Output: Task processing list
1  L ← ∅ /* empty list that will contain the sorted tasks */;
2  S ← $\{i|VP(i) = \emptyset\}$ /* Set of all tasks with no predecessors
     */;
3  while S is not empty do
4      $t \leftarrow \arg\max_i^{|S|} p_J^i$;
5      $S \leftarrow S \backslash \{t\}$          /* remove the task with the maximum
        processing time from S */;
6      $L \leftarrow L + \{t\}$/* add task to tail of L         */;
7      foreach task m with an edge e from t to m do
8          $\mathcal{E}_J \leftarrow \mathcal{E}_J \backslash \{e\}$            /* remove edge e */;
9          if m has no other incoming edges then
10             $S \leftarrow S \cup m$ /* no predecessor          */;
11         end
12     end
13 end
14 if $\mathcal{E}_J$ is not ∅ then
15     return error;
16 else
17     return L;
18 end
```

i5-2520M processor (2 CPUs, 2.5GHz) and 8GB RAM. The optimization problems modeled in section IV was solved using CVXPY (Python Software for Disciplined Convex Programming).

5.1. Baseline approaches

Our approach will be benchmarked against two baseline approaches for task scheduling, namely *ITAGS*[14] and an Offline version of Horae which could be seen as the optimal benchmark for our simulation. We consider each user in the MEC environment only has one job offloaded at any time. Therefore, if there are n users share the MEC environment, there would mostly coexist n different jobs that compete the resources.

- *ITAGS*: Each offloaded job will be scheduled one after the other in

```
Algorithm 4: Handling procedure for new job's arrival
  Input: Job DAG $\langle \mathcal{V}_J, \mathcal{E}_J \rangle$
  Output: Candidate processors set ($P^J$), arrival date ($A^J$), tasks
          processing list ($L^J$) and optimal finish time ($F^J$)
1 Filter out MEC servers which can satisfy placement constrains for
    job $J$ from candidate MEC servers set ;
2 Add new job in unfinished job set;
3 Record the job's arrive time;
4 Generate the task processing list for the job by Alg. 3;
5 Calculate job's optimal finish time ($C^{J(\mathbf{relaxed})}_{monopolize}$) among all MEC
    servers with placement constraints by using eq. (14) ;
6 $F^J \leftarrow A^J + C^{J(\mathbf{relaxed})}_{monopolize}$ ;
```

chronological order.

- *Horae-Offline*: Each offloaded job would be processed simultaneously.

Note that we apply *ITAGS* algorithm scheme with no *placement constraints*, as the system model considered in [14] ignores the placement constraints during offloading.

5.2. Dataset Settings

In the experiments, we vary two setting parameters that may have an impact on the performance of offloading:

- ***Number of offloading users***: With the increasing number of offloading users, especially in crowded CBD area, the number of concurrent offloading service requests would extremely large. For 8 mobile edge servers settings, we experiment various number of concurrent offloading users. To be specific, we enumerate $1, 2, 4, 8, \ldots 64$ concurrent users requesting offloading service.

- ***Number of mobile edge servers***: With the increasing number of available mobile edge servers, the computing capacity of can better meet the offloading demands from mobile users. Therefore, for 10 randomly generated jobs, we experiment various number of available mobile edge servers. To be specific, we calculate $1, 2, 4, 8, \ldots 64$ available mobile edge servers.

Algorithm 5: Horae (online version)

```
while $True$ do
    if $There are unfinished jobs$ then
        if $There comes an available processor p$ then
            $U \leftarrow \emptyset$;
            foreach $unfinished job j$ do
                if $p \in P^j$ then
                    $U \leftarrow U \cup j$ ;
                end
            end
            $h \leftarrow$ the next task of job $j$ to be executed;
            $J^* \leftarrow \arg\max_j^U (F_{est}^p(h,j) - F^j)$;
            Schedule the next task of job $J^*$ on $p$;
            if $h$ is the last task of $j$ then
                Remove $j$ from unfinished jobs;
            end
        end
    end
    if $There comes a new job$ then
        Handling procedure for new job's arrival/* Algorithm 4 */;
    end
end
```

5.3. Metrics

We use two metrics to evaluate both ITAGS and our proposed scheduling method:

- Resource Utilization: The ratio of leveraged resource to the total available resource in MEC environment. Since every job has its fixed resource demand to finish, it is easy to see that the larger resource utilization the smaller response time for job's finish.

- Average Job Finish Time: Since each job's deadline is not same, we have $\alpha = \frac{\text{Job's completion time}}{\text{deadline}}$ to normalize this metric. It is easy to see that the larger α the longer response time for a specific job.

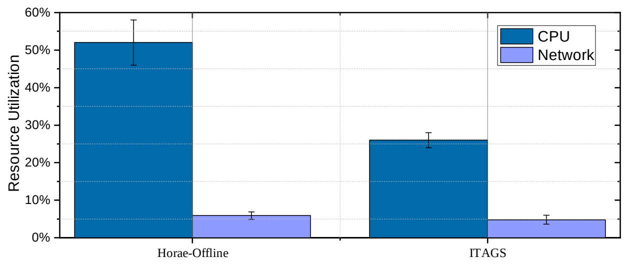


Figure 5: Resource utilization comparison under same MEC environment (10 randomly generated jobs @ 8 parallel edge servers)

5.4. Simulation and results

To compare the average resource utilization between Horae and ITAGS, we conduct simulations based on randomly generated task trees in terms of the DAG structure, task execution times on the processors, and the communication delay between dependent tasks. That is, DAG structure is formed by a collection of vertices and directed edges, each edge connecting one vertex to another, such that there is no way to start at some vertex v and follow a sequence of edges that eventually loops back to v again. A collection of tasks that must be ordered into a sequence, subject to constraints that certain tasks must be performed earlier than others, may be represented as a DAG with a vertex for each task and an edge for each constraint.

In particular, we set by default 8 processors installed on 8 parallel edge servers, and the input/output data amount between each pair of interdependent tasks is drawn uniformly from the interval (1, 3) MB. The communication delay is taken as 10 ns/byte between each processor. We randomly generate 10 different jobs which mostly have 5 interdependent tasks for 100000 times and compare the average resource utilization (calculated by the formula in definition 1) between them over all processors.

From Figure 5, we can easily find that the average network resource utilization is not high because the network resource demand is not strong for computation-intensive application. On the contrary, we can see that Horae got the higher average CPU utilization among the MEC environment because Horae could schedule all offloaded jobs simultaneously by leveraging the spatio-temporal information between them and minimizing the total slack time values of all the jobs.

Considering the fact that network resource is not bottleneck resource among all offloaded jobs, we present comparison only with the average CPU

Table 3: Dataset Settings

Factor	**# of offloading users**	**# of mobile edge servers**
Set #1	$1, 2, 4, 8, \ldots 64$	8
Set #2	10	$1, 2, 4, 8, \ldots 64$

utilization.

Given the dataset settings, we conduct two sets of experiments. The settings are described in Table 3. For each set, we vary one parameter and keep the other one fixed to observe the impact of each parameter on the schemes in the evaluation metrics.

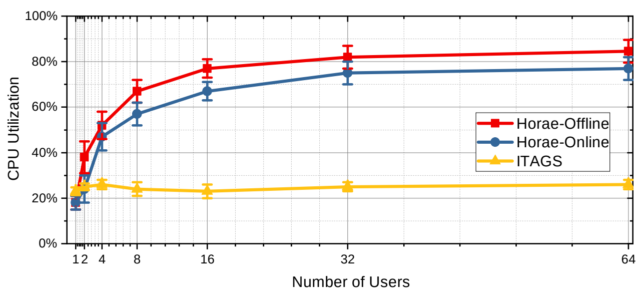
In experiment set #1, the number of offloading users vary from $1, 2, 4, 8$ to 64. The number of mobile edge servers could be leveraged for computation offloading is fixed at 8. In experiment set #2, the number of offloading users fixed at 10. We change the number mobile edge servers that could provide offloading service.

Figure 6 and 7 show the comparison results obtained from the experiment set #1 and #2, respectively. The two performance metrics are depicted in each sub-figure: (a) the Resource Utilization, and (b) Average Job Finish Time.

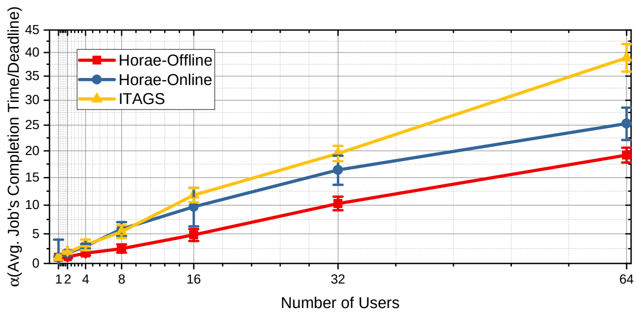
As can be seen in Figure 6(a), the average CPU utilization of Horae-Online and Horae-Offline was almost the same. In addition, Horae scales well according to the number of users because it offers the possibility of performing all the jobs in parallel on the available mobile edge servers, so as the number of users increases, Horae may lead to higher CPU utilization. Conversely, ITAGS performs poorly because it ignores the fact that in a real MEC environment, different users unload multiple tasks simultaneously.

For average job finish time metric, we set the deadline constraint to be $(1 + z) \times C^k_{monopolize}$ for each job k, where $C^k_{monopolize}$ is job k's minimum completion time and can be calculated by exclusively using the whole MEC environment to finish job k. z is a randomly real number following an uniform distribution between 0 and 2, and the mean value of z is set as 1 without specific mention.

Moreover, the jobs are assumed to arrive in a Poisson process, and we set the parameter λ in the Poisson process as $\frac{WirelessNetworkThroughput}{MeanOfInputDataSize}$ default to avoid the whole network overloaded or underloaded, where *MeanOfInput-DataSize* means the data size should be transmitted to trigger the first task of job, and λ can be adjusted to simulate the network load of the environ-



(a) CPU utilization comparison



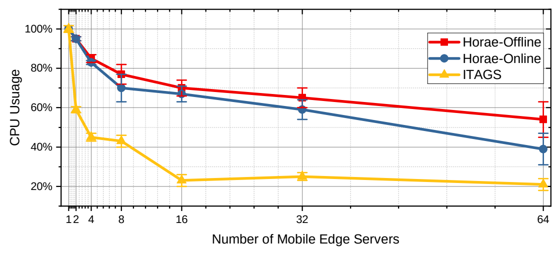
(b) Comparison of job finish time

Figure 6: Resulted metrics of set #1 (number of offloading users changing)

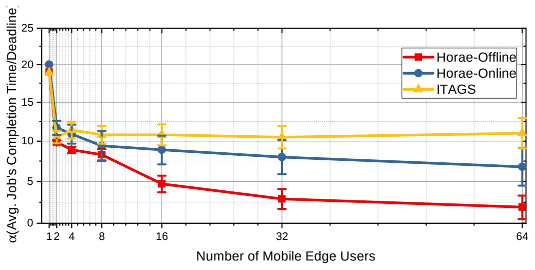
ment. For each set of test parameters, we generate 30 trials to calculate its average and [min, max] range (See errorbars in the Figure 6(b)).

It can be seen from Figure 6(b) that Horae-Online could reduce the average job's finish time, which outperform the performance of ITAGS. The ITAGS performs poorly because it overlooks the fact that multiple jobs could be offloaded simultaneously in a resource-sharing multi-user MEC environment, and ITAGS puts all candidate jobs in a processing queue and schedules each job one after another. Therefore, ITAGS induces the jobs which locate in the rear of the processing queue inevitably waiting a long time to be processed. On the contrary, Horae schedules all candidate jobs simultaneously in the same MEC environment, and try the best to minimize the total *slack time* of all the jobs. Therefore, the average job finish time of horae is smaller than that of ITAGS but we still use the Horae-Offline as the lower bound solution for benchmarking.

In the experiment set #2, we change the number of available mobile edge servers for use. We randomly generate 10 different jobs which mostly have 15 interdependent tasks for 100000 times and compare the average resource



(a) CPU utilization comparison with different number of users



(b) Comparison of job finish time

Figure 7: Resulted metrics of set #2 (number of mobile edge servers changing)

utilization (calculated by the formula in definition 1) between them over all processors. As depicted in Figure 7(a), ITAGS keeps the lower CPU resource utilization comparing to the offline and online version of Horae. Concerning the measure of average task completion time (Figure 7(b)), Horae continued to perform better than ITAGS as the number of moving edge servers increased. Note that ITAGS held steady and had a large average task completion time, especially when the number of edge servers moved exceeded 16. Because ITAGS only schedule one task at a time, and each task mostly has 15 interdependent tasks, when the number of edge servers moving is less than 15, ITAGS can take full advantage of all the machines. However, when the number of mobile edge servers is greater than 15, the ITAGS method will leave the redundant mobile edge servers idle. In contrast, Horae can efficiently leverage all available mobile edge servers for computational offloading, thereby reducing the average job completion time.

6. Summary and Future Work

In this paper, we propose a new offloading scheduler, Horae, in multi-user MEC environment. In contrast to other contemporary outstanding offloading scheduler, Horae takes both placement constraints and precedence constraints of tasks into consideration, and schedules dependent tasks offloaded from different users on heterogeneous processors simultaneously with placement constraints, and therefore improves the resource utilization of the whole MEC environment. The proposed total slack time minimization formulation is generic, allowing different processor topologies and placement constraints. Through randomized simulations, we show the feasibility and efficiency of Horae. In the future, we may consider more complex scenarios and more restrictive conditions. With the continuous development of edge computing, variety of new problems will be found in actual applications, and these are problems that we need to solve in the future.

Acknowledgement

This work is supported in part by the National Natural Science Foundation of China, under Grant No. 61672276, the Key Research and Development Project of Jiangsu Province under Grant No. BE2019104, and the Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing University.

References

[1] S. Wan, Y. Zhao, T. Wang, Z. Gu, Q. H. Abbasi, K. R. Choo, Multi-dimensional data indexing and range query processing via voronoi diagram for internet of things, Future Generation Computer Systems 91 (2019) 382–391.

[2] P. Zeng, B. Pan, K. R. Choo, H. Liu, Mmda: Multidimensional and multidirectional data aggregation for edge computing-enhanced iot, Journal of Systems Architecture 106 (2020) 101713.

[3] Z. Gao, H. Xuan, H. Zhang, S. Wan, K. R. Choo, Adaptive fusion and category-level dictionary learning model for multiview human action recognition, IEEE Internet of Things Journal 6 (6) (2019) 9280–9293.

[4] S. Du, T. Huang, J. Hou, S. Song, Y. Song, Fpga based acceleration of game theory algorithm in edge computing for autonomous driving, Journal of Systems Architecture 93 (2019) 33–39.

[5] J. Feng, L. Zhao, J. Du, X. Chu, F. R. Yu, Energy-efficient resource allocation in fog computing supported iot with min-max fairness guarantees, in: 2018 IEEE International Conference on Communications (ICC), 2018, pp. 1–6. doi:10.1109/ICC.2018.8422317.

[6] S. Wan, Z. Gu, Q. Ni, Cognitive computing and wireless communications on the edge for healthcare service robots, Computer Communications 149 (2020) 99–106.

[7] S. Ding, S. Qu, Y. Xi, S. Wan, A long video caption generation algorithm for big video data retrieval, Future Generation Computer Systems 93 (2019) 583–595.

[8] S. Wan, S. K. Goudos, Faster r-cnn for multi-class fruit detection using a robotic vision system, Computer Networks 168 (2020) 107036.

[9] K. Liu, J. Peng, X. Zhang, Z. Huang, A combinatorial optimization for energy-efficient mobile cloud offloading over cellular networks, in: 2016 IEEE Global Communications Conference (GLOBECOM), 2016, pp. 1–6. doi:10.1109/GLOCOM.2016.7841488.

[10] W. Tang, X. Zhao, W. Rafique, L. Qi, W. Dou, Q. Ni, An offloading method using decentralized p2p-enabled mobile edge servers in edge computing, Journal of Systems Architecture 94 (2019) 1–13.

[11] P. Wang, C. Yao, Z. Zheng, G. Sun, L. Song, Joint task assignment, transmission, and computing resource allocation in multilayer mobile edge computing systems, IEEE Internet of Things Journal 6 (2) (2019) 2872–2884. doi:10.1109/JIOT.2018.2876198.

[12] H. Dai, X. Zeng, Z. Yu, T. Wang, A scheduling algorithm for autonomous driving tasks on mobile edge computing servers, Journal of Systems Architecture 94 (2019) 14–23.

[13] J. Xu, L. Chen, P. Zhou, Joint service caching and task offloading for mobile edge computing in dense networks, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018, pp. 207–215. doi:10.1109/INFOCOM.2018.8485977.

[14] S. Sundar, B. Liang, Offloading dependent tasks with communication delay and deadline constraint, in: IEEE INFOCOM 2018 -

IEEE Conference on Computer Communications, 2018, pp. 37–45. doi:10.1109/INFOCOM.2018.8486305.

[15] G. Zhang, F. Shen, Y. Yang, H. Qian, W. Yao, Fair task offloading among fog nodes in fog computing networks, in: 2018 IEEE International Conference on Communications (ICC), 2018, pp. 1–6. doi:10.1109/ICC.2018.8422316.

[16] J. Du, L. Zhao, J. Feng, X. Chu, Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee, IEEE Transactions on Communications 66 (4) (2018) 1594–1608. doi:10.1109/TCOMM.2017.2787700.

[17] E. Meskar, B. Liang, Fair multi-resource allocation with external resource for mobile edge computing, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2018, pp. 184–189. doi:10.1109/INFCOMW.2018.8406965.

[18] Y. Mao, J. Zhang, K. B. Letaief, Dynamic computation offloading for mobile-edge computing with energy harvesting devices, IEEE Journal on Selected Areas in Communications 34 (12) (2016) 3590–3605. doi:10.1109/JSAC.2016.2611964.

[19] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, I. Stoica, Dominant resource fairness: Fair allocation of multiple resource types., in: Nsdi, Vol. 11, 2011, pp. 24–24.

[20] W. Wang, B. Liang, B. Li, Multi-resource fair allocation in heterogeneous cloud computing systems, IEEE Transactions on Parallel and Distributed Systems 26 (10) (2015) 2822–2835. doi:10.1109/TPDS.2014.2362139.

[21] P. Hunt, M. Konar, F. P. Junqueira, B. Reed, Zookeeper: Wait-free coordination for internet-scale systems., in: USENIX annual technical conference, Vol. 8, Boston, MA, USA, 2010.

[22] E. Cuervo, A. Balasubramanian, D. k. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, Maui: making smartphones last longer with code offload, in: Proceedings of the 8th international conference on Mobile systems, applications, and services, ACM, 2010, pp. 49–62.

[23] H. Zhao, R. Sakellariou, Scheduling multiple dags onto heterogeneous systems, in: Proceedings 20th IEEE International Parallel Distributed Processing Symposium, 2006, pp. 14 pp.–. doi:10.1109/IPDPS.2006.1639387.

[24] L. Özbakir, A. Baykasoğlu, P. Tapkan, Bees algorithm for generalized assignment problem, Applied Mathematics and Computation 215 (11) (2010) 3782–3795.

[25] D. Cattrysse, J. Maes, L. N. Van Wassenhove, Set partitioning and column generation heuristics for capacitated dynamic lotsizing, European Journal of Operational Research 46 (1) (1990) 38–47.

[26] E. L. Lawler, J. K. Lenstra, A. H. R. Kan, D. B. Shmoys, Sequencing and scheduling: Algorithms and complexity, Handbooks in operations research and management science 4 (1993) 445–522.

[27] F. A. Chudak, D. B. Shmoys, Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds, Journal of Algorithms 30 (2) (1999) 323–343.

[28] C. Chekuri, M. Bender, An efficient approximation algorithm for minimizing makespan on uniformly related machines, Journal of Algorithms 41 (2) (2001) 212–224.

[29] A. Rădulescu, A. J. Van Gemund, On the complexity of list scheduling algorithms for distributed-memory systems, in: Proceedings of the 13th international conference on Supercomputing, ACM, 1999, pp. 68–75.

[30] F. A. Potra, S. J. Wright, Interior-point methods, Journal of Computational and Applied Mathematics 124 (1-2) (2000) 281–302.

[31] A. B. Kahn, Topological sorting of large networks, Communications of the ACM 5 (11) (1962) 558–562.

[32] R. E. Tarjan, Edge-disjoint spanning trees and depth-first search, Acta Informatica 6 (2) (1976) 171–185.