Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 202 (2022) 158–163

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Identification, Information and Knowledge in the internet of Things, 2021

Online Task Scheduling Algorithm with Complex Dependencies in Edge Computing

Lei Shi[a], Zhaoxing Ma[a], Yuqi Fan[a], Yi Shi[b], Xu Ding[a], Zhehao Li[a]

[a]*School of Computer Science and Information Engineering, Intelligent Interconnected Systems Laboratory of Anhui Procince, Hefei University of Technology, Hefei 230009, China*

[b]*Virginia Tech, Dept. of ECE, Blacksburg, VA 24061, USA*

Abstract

In the edge computing network environment, our applications can be deployed on edge servers. The request to execute the application can be produced on the edge device and transmitted to the edge server for calculation. A complex request may be divided into multiple computing tasks and transmitted to different servers and then parallel calculated before obtaining the final result. How to schedule computing tasks in multiple requests so that all requests can be completed faster is a difficult problem, especially in the edge computing environment where we should consider the communicating work and the calculating work simultaneously. In this paper, we first build a task dependency model of the computing tasks included in the request based on the idea of dividing the method components of the application. The dependencies include sequence, selection and parallel. Then we propose an online scheduling algorithm MCOS based on optimizing the task with the maximum amount of calculation to solve the problem of the minimum sum of the completion time of all requests. In simulations, we show the algorithm MCOS has a better completion time.

© 2022 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the International Conference on Identification, Information and Knowledge in the Internet of Things, 2021

Keywords: Edge computing; Online scheduling; Dependent task;

1. Introduction

With the development of cloud computing [1, 2], devices with limited computing resources can expand their capabilities by offloading computationally intensive tasks to remote clouds. However, due to the long distance between the device and the remote cloud, severe communication delays will inevitably occur. The situation is even worse when large amount of data needs to be transferred. In order to alleviate this problem, edge computing is proposed as a

* Corresponding author. Tel.: +86-188-5696-4308.
E-mail address: yuqi.fan@hfut.edu.cn

1877-0509 © 2022 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the International Conference on Identification, Information and Knowledge in the Internet of Things, 2021
10.1016/j.procs.2022.04.022

new cloud computing paradigm [3, 4]. By deploying relatively small-scale edge servers at the edge of the Internet, edge computing can provide rich computing resources close to the device, which can significantly reduce propagation delay [5, 6].

Comparing with remote clouds, the resources and the computing power of edge servers are relatively limited [7, 8]. Therefore, multiple edge servers close together often need to cooperate to complete the application. At the same time, modern applications may contain multiple method components with complex data dependencies, and their dependencies can be constructed into a complex set of relationships. In addition, in edge computing, users go online through the device and generate requests to run applications. Although these requests run the same application, the amount of calculation is different due to different input data, so each request can be divided into several computing tasks with dependencies.In order to ensure the overall service quality of the system, so that each user has a good sense of use, it is very important to study how to schedule tasks to each server.

Task scheduling under edge computing is a hot research topic. For example, in [9], authors proposed a hierarchical edge computing shunt framework based on urgent priority, and designed a dynamic priority task scheduling algorithm named as DPTSA, which can reduce the average delay of system tasks, and reduce the delay of high-priority tasks. In [10], Considering the limitations of network bandwidth and computing resources, authors proposed an online algorithm, Dedas, which can greedily schedule newly arrived tasks and consider whether to replace some existing tasks to meet the new deadline. These works are a good solution to the scheduling of a single task without dependency. For tasks with dependencies, many researchers currently only consider parallel relationships, so most of them model the task relationships as directed acyclic graphs (DAG). In [11], authors proposed a novel online algorithm, OnDoc, to solve the problem of on-demand function configuration and DAG scheduling to meet the request deadline as much as possible.

For the scheduling of tasks with more complex dependencies, there is a lack of existing researches. By analyzing the implementation logic of modern applications, we believe that there is still a dependency relationship between priorities and choices between tasks. In this work, we consider the scheduling of tasks with more complex dependencies in edge heterogeneous networks. Our goal is to optimize the overall quality of service. That is, we want to minimize the sum of the completion time for all requests. Our main contributions are as follows:

1) We model tasks with three complex dependencies: *sequence*, *selection* and *parallel* (see Fig. 1).

2) We design a heuristic online scheduling algorithm to minimize the sum of the completion time of all requests as short as possible.

2. System Model and Problem Definition

Consider a two-dimensional network environment consisted with some edge servers (denote as s_i, where $s_i \in \mathcal{N}$) and some devices, see Fig. 1. Suppose each edge server has its own processing speed, and we denote it as f_i for s_i. All edge servers can communicate with each other directly. Denote b_{ij} as the communication rate between s_i and s_j(if $i = j$, $b_{ij} = \infty$). Suppose devices only have the function for collecting data and transmitting them to edge servers for handling. Suppose the transmission rate from devices to edge servers are all the same.

Suppose all collected data need to be handled by a same calculating process, and we call the process as the Application. Suppose each edge server can execute the whole Application alone, or just execute part of the Application, and other parts will be executed by other edge servers. That is, the Application is consisted with several method components, each method component can be handled independently on an edge server. Some dependency relationships may exist in these method components, including *sequence*, *selection* and *parallel* (see Fig. 1). When a device tries to transmit its collected data, it will initiate a request to call the Application on edge servers. Suppose the whole scheduling time can be divided into many time slots τ equally and we normalize $\tau = 1$. Define the time slot set as $\mathcal{T}$. Denote r_k as the request, where $r_k \in \mathcal{R}$. Different requests may have different collected data sizes which will lead to different handling times. When different requests are handled by even a same method component, the input and output data sizes for this method component may also be different. In order to distinguish a method component for these requests, we call each handled process as a computing task. We define computing task as v_i, where $v_i \in \mathcal{V}$. We assume v_{start} as the start computing task, and v_{end} as the end computing task. For each v_i, it can run only when it obtains all input data I_i it needs. Notice that different I_i has different size (which means the number of instructions or the calculation amount of different v_i is different), and the output data O_i will also be different. We define a unique calculation amount function $F_i(I_i)$ for computing task v_i.

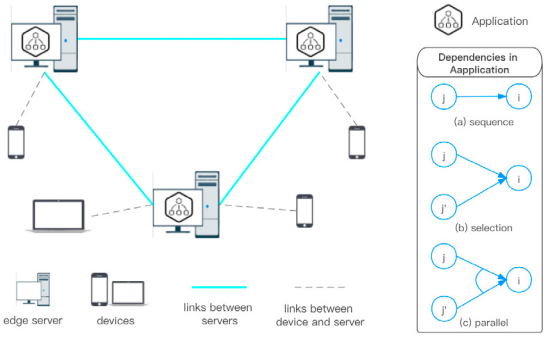

Fig. 1. System Model

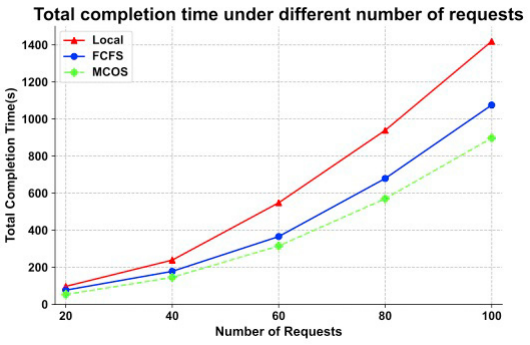

Fig. 2. Total completion time under different number of requests

To sum it up, in our system model, we have many requests with different input data needed to be handled by a same Application. The Application can be divided into many method components and be executed in different edge servers. These method components may have some dependency relationships. And we call a method component for a special request as a computing task. We want to give a scheduling strategy to minimize the sum of the completion times for all requests.

2.1. Task Dependency Model

Define $y_{k,i}(t)$ as the state of computing task $v_i(\in r_k)$ at the t-th time slot, we have

$$y_{k,i}(t) = \begin{cases} 1, & \text{computing task } v_i(\in r_k) \text{ is executed at } t \text{ time slot or } (t=0); \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Now we discuss the expression of the three dependencies.

In Fig. 1(a), v_j points to v_i, which means v_j must be executed before v_i. We call this relationship as *sequence*. In Fig. 1(b), v_i can be executed only by obtaining one of the output data from task v_j or from task $v_{j'}$. We call this relationship as *selection*. In Fig. 1(c), v_i must obtain all output data from task v_j and $v_{j'}$ before it can be executed. We call this relationship as *parallel*. Expressed as follows:

$$\begin{cases} t - t' \geq 1 \\ (y_{k,i}(t) = 1) \wedge (y_{k,j}(t') = 1) \\ (t \neq 0) \wedge (t' \neq 0) \\ v_i, v_j \in r_k. \end{cases}; \begin{cases} t - (t' + t'') \geq 1 \\ (y_{k,i}(t) = 1) \wedge ((y_{k,j}(t') = 1) \vee (y_{k,j'}(t'') = 1)) \\ (t \neq 0) \wedge ((t' \neq 0) \vee (t'' \neq 0)) \\ v_j, v_{j'}, v_i \in r_k. \end{cases}; \begin{cases} t - \max\{t', t''\} \geq 1 \\ (y_{k,i}(t) = 1) \wedge (y_{k,j}(t') = 1) \wedge (y_{k,j'}(t'') = 1) \\ (t \neq 0) \wedge (t' \neq 0) \wedge (t'' \neq 0) \\ v_j, v_{j'}, v_i \in r_k. \end{cases}.$$

Define $\mathcal{Y}_k$ as the set of all these relationship constraints belonging to a request r_k.

2.2. Problem Definition

Define $x_{k,i}(p)$ to show whether task $v_i(\in r_k)$ is handled on the edge server s_p. We have

$$x_{k,i}(p) = \begin{cases} 1, & \text{computing task } v_i(\in r_k) \text{ runs on edge server } s_p; \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Each computing task can only be assigned to one server, so we have

$$\sum_{p \in \mathcal{N}} x_{k,i}(p) \leq 1. \tag{3}$$

If $\sum_{p \in \mathcal{N}} x_{k,i}(p) = 0$, it means task v_i will not run.

We use $\tau_{k,i,p}$ to represent the time length for executing task $v_i(\in r_k)$ on edge server s_p. For a task v_i, the total value of τ is composed with three part. One, the time for receiving data of its previous tasks from the previous edge server. We Denote it as $\tau^{res}_{k,i,p}$. if the previous server and the received server are the same, this value will be zero. Two, the time for waiting to be executed on the received server. We denote it as $\tau^{wat}_{k,i,p}$. Three, the time for executing the task. We denote it as $\tau^{exe}_{k,i,p}$. We have

$$\tau_{k,i,p} = \tau^{res}_{k,i,p} + \tau^{wat}_{k,i,p} + \tau^{exe}_{k,i,p}. \tag{4}$$

In the following we will discuss these three parts separately. Since the time for waiting is the most complex part, we will discuss it at last. We have:

a. Receiving data of the previous tasks. Denote $\tau^{res}_{k,i,p}$ as the time length for receiving the input data of $v_i(\in r_k)$ on the edge server s_p. For the sequence dependency, we have $\tau^{res}_{k,i,p} = x_{k,i}(p)\sum_{p'\in\mathcal{N}}\left(x_{k,j}(p')\frac{O_j}{b_{p'p}}\right)$, where O_j is the output data size for v_j, $b_{p'p}$ is the communication rate between s_p and $s_{p'}$. The selection dependency can also be expressed as sequence. For the parallel dependency, we have $\tau^{res}_{k,i,p} = max\left\{x_{k,i}(p)\sum_{p'\in\mathcal{N}}\left(x_{k,j}(p')\frac{O_j}{b_{p'p}}\right), x_{k,i}(p)\sum_{p''\in\mathcal{N}}\left(x_{k,j'}(p'')\frac{O'_j}{b_{p''p}}\right)\right\}$.

b. Task execution. Denote $\tau^{exe}_{k,i,p}$ as the time length for executing task $v_i(\in r_k)$ on the edge server s_p. We have $\tau^{exe}_{k,i,p} = x_{k,i}(p)\frac{F_i(I_i)}{f_p}$.

c. Waiting for execution. Denote $\tau^{wat}_{k,i,p}$ as the time length for waiting to execute task $v_i(\in r_k)$ on the edge server s_p. $\tau^{wat}_{k,i,p}$ is from the time slot where the received data has reached the edge server totally to the time slot where the execution is started. Since we know for task v_i, t is the start time slot when $y_{k,i}(t) = 1$ and $t \neq 0$, so if we can represent the time slot for receiving, then we can represent $\tau^{wat}_{k,i,p}$. Notice that the time slot for receiving can be calculated by two elements. One, the time slot for starting the execution of the previous tasks. Two, the time length for executing the previous tasks and the time length for transmitting the input data for v_i. Based on these, we have

$$Sequence: \begin{cases} \tau^{wat}_{k,i,p} = x_{k,i}(p)(t - t' - \tau^{exe}_{k,j,p'} - \tau^{res}_{k,i,p}) \\ (y_{k,i}(t) = 1) \wedge (y_{k,j}(t') = 1) \\ (t \neq 0) \wedge (t' \neq 0) \\ v_i, v_j \in r_k. \end{cases}; \quad Selection: \begin{cases} \tau^{wat}_{k,i,p} = x_{k,i}(p)(t - (t' + t'') - (\tau^{exe}_{k,j,p'} + \tau^{exe}_{k,j',p''}) - \tau^{res}_{k,i,p}) \\ (y_{k,i}(t) = 1) \wedge ((y_{k,j}(t') = 1) \vee (y_{k,j'}(t'') = 1)) \\ (t \neq 0) \wedge ((t' \neq 0) \vee (t'' \neq 0)) \\ v_j, v_{j'}, v_i \in r_k. \end{cases};$$

$$Parallel: \begin{cases} \tau^{wat}_{k,i,p} = \max\left\{x_{k,i}(p)(t - t' - \tau^{exe}_{k,j,p'} - \tau^{res}_{k,i,p}), x_{k,i}(p)(t - t'' - \tau^{exe}_{k,j,p''} - \tau^{res}_{k,i,p})\right\} \\ (y_{k,i}(t) = 1) \wedge (y_{k,j}(t') = 1) \wedge (y_{k,j'}(t'') = 1) \\ (t \neq 0) \wedge (t' \neq 0) \wedge (t'' \neq 0) \\ v_j, v_{j'}, v_i \in r_k. \end{cases}.$$

We know that each request r_k has only one start task v_{start} and one end task v_{end}. We denote their executed time slot as t_{start} and t_{end}, respectively. Our optimization goal is to minimize the sum of the completion time of all requests.

$$\begin{aligned} \min: \ & \textstyle\sum_{k\in|\mathcal{R}|}(t_{end} - t_{start} + \tau^{exe}_{k,end,p}) \\ s.t. \quad & y_{k,start}(t_{start}) = 1 \qquad v_{start} \in r_k \\ & y_{k,end}(t_{end}) = 1 \text{ and } t_{k,end} \neq 0 \ \ v_{end} \in r_k \\ & \tau^{wat}_{k,end,p} \geq 0 \end{aligned} \tag{5}$$

3. Algorithms

We describe the details of our algorithm in this section. Through the description of the system model, we know that the arrival time and initial data volume of each request are different, which makes the difference between all computing tasks. Therefore, it is difficult for us to uniformly allocate some tasks as a class of tasks. The manifestation of this difference is the impact of the amount of input data on the amount of calculation of the task. Therefore, the design of our algorithm part mainly focuses on the analysis and construction of the amount of calculation of the task. We propose an online task scheduling algorithm based on the maximum amount of calculation(MCOS), which is divided into three parts. We first construct the priority between the computing tasks included in a single request by simplifying the task dependency model in sub section 3.1. Then in sub section 3.2, we design an algorithm to distribute computing tasks to edge servers. Finally, we give the edge server queue optimization algorithm in sub section 3.3.

3.1. Streamlined Task Dependency Model

The calculation amount of a single calculation task is affected by the amount of input data and is computable. So when the initial amount of data for a request is determined, we can in principle calculate the amount of calculation for all tasks. However, due to the selected task dependency, there are many possibilities for the calculation of subsequent tasks. Therefore, when a request arrives, we streamline the selected task dependency based on its initial data volume. The principle of simplification is to select the computing task that minimizes the calculation amount of the entire request. We use l_k to represent the list of computing tasks to be run after request r_k has been streamlined. We define all lists as set Q, where $l_k \in Q$. At the same time, the total amount of calculation for a single request is also an important factor for us to assess the priority between requests. This will be reflected in the algorithm design section below.

3.2. Computing Task Dispatch

In this part, we will propose an algorithm to select the appropriate edge server for dispatch for each computing task. By analyzing our system model, we can obtain the current queue of each edge server and complete the time slot sequence number of the current queue. This allows us to remove some non-compliant edge servers, and for the remaining candidate edge servers, we choose the one that makes the calculation task complete the earliest. The specific algorithm design is shown in Algorithm 1, with the following detailed steps:

Step 1: We first judge whether there is a new request coming. If a new request r_k comes, we will streamline it according to the streamlining principle and build a computing task list l_k. Then add the list l_k to set Q.(Line 3-6)

Step 2: According to the set Q, we can know the completion progress of each request in the current time slot, and we can construct a computing task queue H to be dispatched under the current time slot. Then, we sort the queue H according to the calculation amount of the computing task from large to small.(Line 7-8)

Step 3: For each computing task to be dispatched, we construct a set of selectable edge servers $N_{k,i}$ according to the current network situation. If the current task queue completion time slot number of an edge server is less than the average transmission time $\frac{1}{|\mathcal{N}|}\sum_{p\in\mathcal{N}}\tau^{res}_{k,i,p}$ of the current task to be dispatched, it can be added to the set $N_{k,i}$. If there is no edge server that meets the requirements, select the two edge servers with the earliest completion time. Then select the edge server $s_p \in N_{k,i}$ to minimize the current computing task completion time $\tau_{k,i,p}$ for dispatch.(Line 9-20)

Algorithm 1 Computing Task Dispatch Algorithm

Input: Initializing $\mathcal{N}$, $\mathcal{M}$, b, B, $\mathcal{V}$, $\mathcal{Y}$, $\mathcal{R}$;
1: set $Q \leftarrow \varnothing$;
2: **for** *each time slot in T* **do**
3: **if** *new request r_k arrives* **then**
4: $l_k \leftarrow$ Build a computing task list form r_k;
5: $Q \leftarrow Q \cup \{l_k\}$;
6: **end if**
7: Construct queue H according to the set Q ;
8: Sort queue H according to the calculation amount of the computing task from large to small;
9: **for** *each computing task($v_i \in r_k$) in H* **do**
10: set $N_{k,i} \leftarrow \varnothing$;
11: **for** *each edge server $s_p \in \mathcal{N}$* **do**
12: **if** completion time of $s_p < \frac{1}{|\mathcal{N}|}\sum_{p\in\mathcal{N}}\tau^{res}_{k,i,p}$ **then**
13: $N_{k,i} \leftarrow N_{k,i} \cup \{s_p\}$;
14: **end if**
15: **end for**
16: **if** $N_{k,i}$ is $\varnothing$ **then**
17: Select two edge servers with the earliest completion time and add them to $N_{k,i}$;
18: **end if**
19: Select the edge server $s_p \in N_{k,i}$ to minimize $\tau_{k,i,p}$ for dispatch;
20: **end for**
21: **end for**

Algorithm 2 Computing Task Queue Optimization Algorithm

Input: Algorithm 1 dispatch result;
1: **for** *each time slot in T* **do**
2: **if** *new computing task $v_i \in r_k$ is dispatched to edge server s_p* **then**
3: Construct the task queue q_{s_p} of the edge server s_p;
4: **for** *each computing task $v_{i'} \in r_{k'}$ in q_{s_p}* **do**
5: **if** calculation amount of r_k is greater than $r_{k'}$ **then**
6: Move the computing task $v_i \in r_k$ before $v_{i'} \in r_{k'}$;
7: **end if**
8: **end for**
9: **end if**
10: **end for**

3.3. Computing Task Queue Optimization

Through Algorithm 1, we dispatched each computing task to a suitable edge server, which is in line with our optimization goal. After further analysis, we can know that the total amount of calculation for different requests is different. Obviously, the faster the request with a large amount of total calculation is completed, the more our optimization goal is met. On this basis, we propose a computational task queue optimization algorithm. The specific algorithm design is shown in Algorithm 2, with the following detailed steps:

Step 1: When a new computing task $v_i \in r_k$ is dispatched to the edge server s_p through algorithm 1, we first construct a computing task queue q_{s_p} of the current server s_p to be run.(Line 3)

Step 2: Then we compare the calculation amount of the request r_k to which the computing task v_i belongs to the calculation amount of the request to which the calculation task belongs in the queue p. Then we compare the calculation amount including the computing task v_i request r_k and the calculation amount of the request belonging to the computing task in the queue q_{s_p}. If the calculation amount of a certain request r_k is larger, move v_i before the computing task.(Line 4-7)

4. Simulation

In this section, we will present the simulation results. We first constructed an application composed of some method components. Then we specify the size of our single time slot τ as 10 ms. On this basis, we set up the network environment. We set the number of our edge servers to five, and the processing speed of each edge server is different, and the value ranges from 4 to 8 GHz. The communication rate between any two edge servers is from 1 to 5 MB/s. For each request, we set its initial data size between 500 and 1000 KB, and the time to reach the edge network is random. We will compare our algorithm with the two baselines by changing the number of requests. Due to the lack of research on the assignment of the three dependency tasks we described, we take two typical algorithms as our baselines. **Local Heuristic (Local)**: When a request reaches its initial receiving server, all computing tasks of this request will be run on this edge server. **First-Come-First-Serve (FCFS)**: FCFS is a popular scheduling policy which is commonly used by the methods based on queuing theory [12].

We use the arrival time of the request as the queuing order, and the next request will be allocated after all the computing tasks of a request have been allocated. We first randomize the processing speed of five edge servers and the communication rate between them. Then we use 20 requests as an incremental basis set, and randomly get their initial data volume and arrival time, and then compare the completion times of the three algorithms, the result is shown in Figure. 2. Since the number of our edge servers is 5, the pros and cons of the three algorithms are not obvious when the number of requests is 20. But as the number of requests increases, our algorithm MCOS is always the lowest in the sum of the completion time. And the gap with the other two algorithms is gradually increasing.

5. Conclusion

In this work, we study how an edge server network that deploys applications with sequence, selective, and parallel relationships can schedule random online requests with different initial data. We constructed a general model for this problem to minimize the completion time of all requests. We give an online task scheduling algorithm MCOS, which is the first algorithm used to solve online scheduling applications with sequence, selective and parallel relations. Compared with the two baseline algorithms, MCOS has a better completion time.

References

[1] B. Chun, Sunghwan Ihm, Petros Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *EuroSys '11*, 2011.

[2] Yangming Zhao, Xin Liu, and C. Qiao. Job scheduling for acceleration systems in cloud computing. *2018 IEEE International Conference on Communications (ICC)*, pages 1–6, 2018.

[3] P. López, A. Montresor, D. Epema, Anwitaman Datta, T. Higashino, A. Iamnitchi, Marinho P. Barcellos, P. Felber, and E. Riviere. Edge-centric computing: Vision and challenges. *Comput. Commun. Rev.*, 45:37–42, 2015.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[6] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys Tutorials*, 19(4):2322–2358, 2017.

[7] J. Ren, G. Yu, Y. Cai, and Y. He. Latency optimization for resource allocation in mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 17(8):5506–5519, 2018.

[8] C. You, K. Huang, H. Chae, and B. Kim. Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 16(3):1397–1411, 2017.

[9] J. X. Liao and X. W. Wu. Resource allocation and task scheduling scheme in priority-based hierarchical edge computing system. In *2020 19th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, pages 46–49, 2020.

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

[11] Liuyan Liu, Haoqiang Huang, Haisheng Tan, Wanli Cao, Panlong Yang, and Xiang-Yang Li. Online dag scheduling with on-demand function configuration in edge computing. In Edoardo S. Biagioni, Yao Zheng, and Siyao Cheng, editors, *Wireless Algorithms, Systems, and Applications*, pages 213–224, Cham, 2019. Springer International Publishing.

[12] Haisheng Tan, Zhenhua Han, X. Li, and F. Lau. Online job dispatching and scheduling in edge-clouds. *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017.