Hindawi
Security and Communication Networks
Volume 2022, Article ID 6430612, 14 pages
https://doi.org/10.1155/2022/6430612

Research Article

PACAM: A Pairwise-Allocated Strategy and Capability Average Matrix-Based Task Scheduling Approach for Edge Computing

Feng Hong, Tianming Zhang, Bin Cao, and Jing Fan

College of Computer and Science College of Software, Zhejiang University of Technology, Hangzhou, China

Correspondence should be addressed to Tianming Zhang; tmzhang@zjut.edu.cn

Received 22 September 2021; Revised 25 November 2021; Accepted 8 December 2021; Published 11 January 2022

Academic Editor: Yuyu Yin

With the development of the smart Internet of Things (IoT), an increasing number of tasks are deployed on the edge of the network. Considering the substantially limited processing capability of IoT devices, task scheduling as an effective solution offers low latency and flexible computation to improve the system performance and increase the quality of services. However, limited computing resources make it challenging to assign the right tasks to the right devices at the edge of the network. To this end, we propose a polynomial-time solution, which consists of three steps, i.e., identifying available devices, estimating device quantity, and searching for feasible schedules. In order to shrink the number of potential schedules, we present a pairwise-allocated strategy (PA). Based on these, a capability average matrix (CAM)-based index is designed to further boost efficiency. In addition, we evaluate the schedules by the technique for order preference by similarity to an ideal solution (TOPSIS). Extensive experimental evaluation using both real and synthetic datasets demonstrates the efficiency and effectiveness of our proposed approach.

1. Introduction

As an essential function, task scheduling of edge computing has widespread applications in various domains such as the Internet of vehicle [1], transportation [2], health emergency [3], and smart homes [4]. With the development of the smart Internet of Things (IoT) [5–7], an increasing number of tasks deployed on the edge of the network and the IoT devices require to process the heavy tasks with finite response time. Considering the substantially limited processing capability, how to assign tasks to these devices is a significant issue to improve the performance of the system and increase the quality of services. In the following, we give a representative example.

1.1. Motivation Example. (*Internet of vehicle*) Unmanned aerial vehicles (UAVs) are one type of new IoT devices that are usually used for surveying and mapping services. The number of tasks containing surveying and mapping is large, and if most tasks are assigned to a small part of UAVs, this causes these UAVs to require much time to process tasks and other UAVs do a few tasks, even none. The time cost of services depends on the time point of finishing the last task, and such assignment making not full use of devices results in a longer response time of finishing a service. In addition, considering limited capability, UAVs are unable to execute the heavy tasks, which results in low quality of services since a few tasks are not finished. Given these limitations, an effective and efficient task scheduling helps to assign the right tasks to the right UAVs and reduce the time cost of finishing services on the premise of guaranteeing the quality of services.

In past years, important research focused on the mathematically strict scheduling models and effective techniques for task scheduling problems of edge computing [8–10]. Few works consider the optimization goals in aspects of fairness; that is, most works consider the global result of each optimization goal in the whole schedule while ignoring its changing local results [11]. Besides, it is well known that task scheduling is an NP-hard problem [12]. Some existing works adopted mathematical programming, such as mixed-Integer programming (MIP [13–17]). Each of them ordinarily adopts the solvers to generate the feasible schedules, and this process requires prohibitive computations to be supported.

In addition, there are many similar studies on task scheduling problems of edge computing [17–18], and they may suffer from one or more of the following drawbacks:

(1) Most of the studies model their problems and adopt a series of pruning techniques to shrink the number of potential schedules, such as substituting the precise solutions with approximate ones. Although this way can reduce a part of the computation burden, it remains to be time-consuming.

(2) The task scheduling problems usually involve more than one optimization goal, and the potential conflicts between objectives may exist; i.e., the improvement of one goal may lead to the performance degradation of another. Many studies convert multiple goals to one single objective by empirically using different weights over different goals. However, too much domain knowledge is needed in this way and the generated schedule may fail if the weights are changed.

In our defined task scheduling problem, two categories of constraints are defined. (1) Hard constraint: we consider two hard constraints; i.e., *each device is assigned to at most one task per service* and *each device works at most k consecutive services*. Any one of them is broken will lead to the invalidity of the schedule. (2) Soft constraint: that is, *the total capabilities of the assigned devices for each type of task are equal to its workload*, which should be maximally satisfied. To effectively evaluate the quality of a schedule, we define two optimization goals, i.e., the average coverage *Ave_Coverage* and the fairness of coverage *Coverage_Fairness*, where *Ave_Coverage* computes the coverage ratio of the whole schedule and *Coverage_Fairness* is utilized to measure that all the coverage ratios in the schedule get close to 1. The closer *Ave_Coverage* is to 1 and the smaller *Coverage_Fairness* is, the higher the quality of the schedule will be.

Based on these, we propose a pairwise-allocated strategy and capability average matrix-based approach (PACAM), which is a polynomial solution. To be more specific, PACAM consists of three steps, i.e., identifying available devices, estimating device quantity, and searching for a feasible schedule. In the first step, for each device, PACAM checks whether it is available for tasks according to hard constraints. If one device broke any one of the hard constraints on one service, it will be identified to be unavailable and cannot be assigned to tasks on this service. Next, we need to assign them to tasks. When assigning available devices to tasks directly, we will face a large number of potential different device combinations, which contain plenty of obviously unfeasible device combinations; e.g., all the devices are assigned to the same type of tasks. Hence, in the second step, PACAM preestimates the number of devices for each type of task by the workload for each type of task and the average capability of devices. Then, in the third step, PACAM assigns the corresponding estimated number of devices to each type of task, which effectively reduces the potential device combinations by pruning obviously unfeasible device combinations in advance.

However, evaluating each device combination for searching for the feasible schedule after the pruning technique still requires a great amount of computational consumption; thus, PACAM proposes the pairwise-allocated strategy (PA) to solve this problem. PA treats two devices as a device group and assigns device groups to the corresponding task according to their average capability. Note that the number of devices in these device groups for each task is equal to its estimated number. The reason is that the estimated number of devices for each task is computed by the workload of each task and average capability of devices, the closer the average capability of these device groups is to that of all the devices, the smaller the difference between the total capabilities of these device groups and the workload of the corresponding tasks is. To boost its efficiency, PACAM proposes the capability average matrix (CAM). Besides, PACAM introduces the technique for order preference by similarity to an ideal solution (TOPSIS, [19]) as the evaluation metric. The reason why we use TOPSIS is that it is a multicriteria decision analysis method [20] and is widely used in evaluating a solution with different dimensions. TOPSIS provides enough information for PACAM to generate a wise feasible schedule.

We experimentally evaluate PACAM using a variety of real and synthetic datasets coming from China Telecom company. The real part of datasets contains daily call arrivals for six months of 2020, and the synthetic part is the devices and their capabilities in the corresponding months. We demonstrate the effectiveness and efficiency of PACAM on these datasets. It only takes a few minutes to search for a feasible schedule. In general, the contributions in this article are summarized as follows:

(1) We propose a polynomial-time solution to efficiently handle task scheduling problems.

(2) We present PACAM, where a pairwise-allocated (PA) strategy is proposed to shrink the number of potential device combinations and a CAM index is designed to boost efficiency further.

(3) We first introduce the TOPSIS to effectively evaluate the potential schedules and decide the final feasible schedule.

(4) Using real-life data (daily call arrivals in 6 months of 2020) and synthetic data (capabilities of devices), we experimentally verify the effectiveness and efficiency of our proposed method. Compared with five state-of-the-art methods for task scheduling, i.e., linear programming (LP [21]), integer programming (IP [22]), MIP [16]), improved particle swarm optimization (IPSO [23]), and multiobjective evolutionary algorithm-based decomposition (MOEAD [24]), we find that PACAM is at least two orders of magnitude faster than the above methods.

The rest of this article is organized as follows. Section 2 introduces several concepts and optimization goals and defines the problem formally. Section 3 elaborates a polynomial-time solution for task scheduling. Experimental

results and our findings are reported in Section 4. Section 5 reviews related work. Finally, Section 6 concludes the article with some directions for future work.

2. Preliminaries

In this section, first, we present a series of basic concepts, i.e., scheduling horizon, devices, and scheduling constraints. Then, we introduce two optimization goals. Finally, we define the task scheduling problem. For ease of reference, Table 1 summarizes the notations used frequently in this article.

2.1. Basic Concepts. We start with notations of each basic concept.

2.1.1. Scheduling Horizon. A scheduling horizon is defined as $S = \{s_1, s_2, \ldots, s_n\}$, where $s_i \in S$ represents a service. Each service s_i has two types of tasks, i.e., day task and night task, denoted as DT_i and NT_i, respectively. In addition, each type of task for s_i has a corresponding workload. Specifically, the workloads of the day task DT_i and night task NT_i are represented as WD_i and WN_i, respectively.

2.1.2. Devices. Devices are defined as $D = \{d_1, d_2, \ldots, d_m\}$, where each device $d_j \in D$ has a capability c_j. After devices are assigned to tasks, the total capabilities of the devices assigned to the day task DT_i and the night task NT_i are denoted as $totalc\_DT_i$ and $totalc\_NT_i$, respectively.

It is worth noting that the workload corresponds to the device's capabilities. For example, a device whose capability is 2 is assigned to the day task, which means that the workload of this task this device finishes is 2.

2.1.3. Scheduling Constraints. When assigning devices to tasks, constraints such as devices' maintenance, business rules, and regulations are considered. They are divided into two types, i.e., hard constraints that can not be violated and soft constraint that should be maximally satisfied.

In this article, we mainly consider two hard constraints and one soft constraint. Specifically, hard constraints include the following:

(1) $Hard_1$, each device is assigned to at most one type of tasks per service.

(2) $Hard_2$, each device works at most k consecutive services.

The soft constraint is as follows:

(1) $Soft_1$, the total capabilities of the assigned devices for each type of task are equal to its workload.

2.2. Optimization Goals. Following the principle that soft constraints should be maximally satisfied, we proposed two optimization goals: *average workload coverage* and *coverage fairness*.

2.2.1. Average Workload Coverage. A feasible assignment should follow that the total capabilities of the assigned devices maximally satisfy the workloads of tasks. Hence, we define *Ave_Coverage* to compute the coverage ratio of the whole schedule.

Given a scheduling horizon $S = \{s_1, s_2, \ldots, s_n\}$ and a set of devices $D = \{d_1, d_2, \ldots, d_m\}$, devices are assigned to the day and night tasks, the *Ave_Coverage* is computed by

$$\text{Ave\_Coverage} = \frac{1}{2n} \times \sum_{i=1}^{n} \left(\frac{\text{ac\_DT}_i}{\text{WD}_i} + \frac{\text{ac\_NT}_i}{\text{WN}_i} \right). \quad (1)$$

Example 1. The workload of the three services is listed in Table 2 and the total capabilities for each type of task of each service are shown in Table 3. To compute Ave_Coverage, thus, according to equation (1), *Ave_Coverage* = (1/6)× ((2.2/2.2) + (1.1/ 1.0) + (1.2/0.8) + (2.3/4.6) + (2.3/2.3)+ (2.3/2.3)) = 1.02.

Note that *Ave_Coverage* is used to compute the total average coverage ratio of the whole schedule, but the coverage ratios of some types of tasks are not close to 1; e.g., the coverage ratios of DT_2 and NT_2 are 1.5 and 0.5, which violate $Soft_1$. Hence, we adopt *Coverage_Fairness* to ensure that the coverage ratio gets close to 1.

2.2.2. Coverage Fairness. Given a scheduling horizon $S = \{s_1, s_2, \ldots, s_n\}$ and a set of devices $D = \{d_1, d_2, \ldots, d_m\}$, the *Coverage_Fairness* is defined as follows:

$$\text{Coverage\_Fairness} = \left\{ \frac{1}{2n} \times \sum_{i=1}^{n} \left[\left(\frac{\text{ac\_DT}_i}{\text{WD}_i} - \text{Ave\_Coverage} \right)^2 + \left(\frac{\text{ac\_NT}_i}{\text{WN}_i} - \text{Ave\_Coverage} \right)^2 \right] \right\}^{1/2}. \quad (2)$$

Consider Example 1, according to equation (2), *Coverage_Fairness* = $\{(1/6) \times [(1 - 1.02)^2 + (1.1 - 1.02)^2 + (1.5 - 1.02)^2 + (0.5 - 1.02)^2 + (1 - 1.02)^2 + (1 - 1.02)^2]\}^{1/2} = 0.289$.

2.3. Problem Definition. Based on the above concepts, we define the problem formally.

2.3.1. Task Scheduling Problem. Given a scheduling horizon $S = \{s_1, s_2, \ldots s_n\}$ and a set of devices $D = \{d_1, d_2, \ldots, d_m\}$, the task scheduling aims to assign the devices in D to the tasks of services in S, such that

(1) hard constrains $Hard_1$ and $Hard_2$ must be satisfied

(2) | 1-*Ave_Coverage* | is minimized

Table 1: Symbols and description.

Annotation	Description
$S = \{s_1, s_2, \ldots, s_n\}$	A scheduling horizon of n services
DT_i or NT_i	The day or night tasks of the service s_i
WD_i or WN_i	The workload of the day task DT_i or the night task NT_i
$D = \{d_1, d_2, \ldots, d_m\}$	A set of m devices
$C = \{c_1, c_2, \ldots, c_m\}$	The capabilities set of m devices
$Hard_1$, $Hard_2$	The hard constraints
$Soft_1$	The soft constraint
ac_DT_i / ac_Nt_i	The total capabilities of devices assigned to the day task DT_i or the night task NT_i
total_AC	The total number of capabilities of available devices
k	The maximal number of consecutive services
available_D	The set of devices that can be assigned to tasks
$(task\_s_{i-k+1}, \ldots, task\_s_i)$	The previous k tasks of s_i
$number\_DT_i$/$number\_NT_i$	The estimated number of devices for the day/night task of s_i
$totalac\_s_i$	The total quantity of capabilities that can be assigned to tasks of s_i
DC_DT_i/DC_NT_i	The device combinations for the day and night tasks of s_i

Table 2: The workload for each type of task of each service.

Horizon	WD_i	WN_i
s_1	2.2	1.0
s_2	0.8	4.6
s_3	2.3	2.3

Table 3: The assigned capability of each type of task.

Horizon	Total assigned capability	
	ac_DT_i	ac_NT_i
s_1	2.2	1.1
s_2	1.2	2.3
s_3	2.3	2.3

(3) the value of *Coverage_Fairness* is minimized

3. Algorithm

A naive way to find the optimal schedule is to first enumerate all the possible device combinations and then choose the best one that minimizes the values of |1 − Ave_Coverage| and *Coverage_Fairness*. However, it is computationally intractable because the number of possible device combinations is exponential (i.e., $|S| \times 2^{|D|}$), where $|S|$ and $|D|$ are cardinalities of the scheduling horizon S and the set D of device. To reduce the asymptotic complexity of device scheduling, we design a PA strategy that dramatically shrinks the number of potential device combinations and defines a CAM index to boost efficiency. Based on these, a polynomial-time solution is proposed. The pseudocode is shown in Algorithm 1.

Specifically, we arrange the tasks by services (line 1); each service is composed of three procedures, namely, identifying available devices *identify_device*() (line 2), estimating device quantity *estimate_quantity* (lines 3-4), and searching for feasible schedule *search_schedule* (line 5). Device (D) will be input into *identify_device*() to generate a set of available devices (*available_D*). Then, *available_E* and the workload of each type of task will be computed by *estimate_quantity*() to estimate the number of devices for each type of task of the service s_i (the estimated quantity of devices for day/night tasks $number\_DT_i$ / $number\_NT_i$). Next, *search_schedule* generates the feasible schedule according to the workload for each type of task of the service s_i (the workload of day/night task WD_i / WN_i) and the estimated quantity of devices for each type of task. In the following, we detail these three key procedures.

3.1. Identifying Available Device. For the service s_i ($k \le i < n$), not all the devices can be assigned to the tasks of s_i, since the hard constraint $Hard_2$ rules that the maximal number of consecutive services cannot exceed k services for a device. Hence, for each device, the procedure *identify_device*() checks whether there is a Rest-status in its tasks of the previous k services (i.e., $s_{i-k+1}, s_{i-k+2}, \ldots, s_k$). If the answer is yes, the device is identified to be available for s_i. As for the service s_i ($0 < i < k$), all devices are identified to be available, since even if the device d_j is assigned to tasks during the previous i services, the maximal consecutive services of d_j for s_i cannot be exceed k services.

Next, we give an example to show how we identify available devices, combined with the procedure of *identify_device*(), which is presented in Algorithm 2.

Example 2. Consider a scheduling example as shown in Table 4, where there are 7 devices and all of them have been assigned to tasks of s_1 and s_2, suppose that the maximal number of consecutive services k is set to 2; now we need to identify the availability of devices for the service s_3. From Table 4, since the service s_3 ($k < 3 < n$, $k = 2$), we get tasks of the previous 2 services for each device (line 3), then we traverse the tasks of each device to check for whether they contain a Rest-status (line 4). In this example, the tasks of the devices d_1, d_2, d_3, d_4, d_5 contain a Rest-status and the others are not. The device with the tasks containing a Rest-status will be identified to be available and be stored in the set of available devices available_D (line 5). Thus, the devices d_1, d_2, d_3, d_4, d_5 will be put into available_D, and the devices d_6, d_7 are

Input: The set of devices **D**; the workload WD_i and WN_i; the maximal consecutive services **k**
Output: The feasible schedule **fs**
(1) **for** *each service* s_i *in* S **do**
(2) *available_D* ← **identify_device** (D, k, s_i)
(3) *number*_DT_i ← **estimate_quantity** (WD_i, *available_D*)
(4) number_NT_i ← **estimate_quantity** (WN_i, *available_D*)
(5) *fs* ← **search_schedule** (WD_i, WN_i, *number*_DT_i, *number*_NT_i)
(6) **Return** *fs*

Algorithm 1: The overview of PACAM.

Input: The set of device **D**; the maximal consecutive services **k**; the i^{th} service s_i
Output: The set of available devices **available_D** s_i)
(1) **for** each service s_i ($k < i < n$) **do**
(2) **for** each device d_j in D **do**
(3) (task_s_{i-k+1}, task_s_{i-k+2}, ..., task_s_k) ← get tasks of the previous k services for d_j;
(4) **if** (task_s_{i-k+1}, task_s_{i-k+2}, ..., task_s_k) contains a Rest-status **then**
(5) put d_j into *avaiable_D*;
(6) **for** each service s_i ($0 < i \leq k$) **do**
(7) **for** each device d_j **do**
(8) put d_j into *avaiable_D*;
(9) **Return** *available_D*

Algorithm 2: identify_device (D, k, s_i).

Table 4: The assigned capability of each type of task.

Device	Service s_1	Service s_2	Service s_3
d_1	Day task	Rest-status	Available
d_2	Night task	Rest-status	Available
d_3	Rest-status	Day task	Available
d_4	Rest-status	Day task	Available
d_5	Rest-status	Rest-status	Available
d_6	Day task	Night task	Unavailable
d_7	Night task	Day task	Unavailable

identified to be unavailable for the service s_3. As for the service s_i ($0 < i \leq k$, $k = 2$), each device is available and will be stored in available_D (lines 6–8).

3.2. Estimating Device Quantity. After the available device of s_i is identified, we need to assign them to the tasks of s_i. To avoid enumerating the exponential device combinations, we invoke the procedure *estimate_quantity*() (as Algorithm 3 shows) to estimate the number of devices required for each type of task in advance. Specifically, Algorithm 3 takes the set of available devices and the workload of each type of task of s_i as inputs, and the output is the estimated quantity of available devices required for each type of task of s_i.

First, according to the available device set, we compute their total capability *total_AC* and average capability *average_AC* (lines 1-2). Based on this, we estimate the number of devices required for each type of task. To avoid the understaffing/overstaffing problem, we need to confirm the total quantity of available capability totalac_s_i by the following equation:

$$\text{totalac\_}s_i = \begin{cases} \text{total\_AC}, & \text{total\_AC} < \text{WD}_i + \text{WN}_i, \\ \text{WD}_i + \text{WN}_i, & \text{total\_AC} \geq \text{WD}_i + \text{WN}_i, \end{cases} \quad (3)$$

where totalac_s_i is the total quantity of capabilities that can be assigned to tasks of s_i, *total_AC* denotes the total number of capabilities of all available devices of s_i, and WD_i/WN_i represents the workload of the day/night task of s_i. If *total_AC* $< WD_i$+WN_i, we assign all available devices to tasks and *totalac_*s_i = *total_AC* (lines 3 and 4). Otherwise, part of available devices will be assigned to tasks and totalac_s_i WD_i+WN_i (lines 5 and 6).

Then based on the ratio of the workload of each task to that of all tasks, we compute the estimated quantity of each type of task, shown in the following equation (lines 7-8):

$$\begin{cases} \text{ac\_DT}_i = \text{totalac\_}s_i \times \dfrac{\text{WD}_i}{\text{WD}_i+\text{WN}_i}, \\ \text{ac\_NT}_i = \text{totalac\_}s_i \times \dfrac{\text{WN}_i}{\text{WD}_i+\text{WN}_i}, \end{cases} \quad (4)$$

where ac_DT_i and ac_NT_i are the number of capabilities that can be assigned to the day and night tasks of s_i, totalac_s_i is the total quantity of capabilities that can be assigned to tasks of s_i, and WD_i/WN_i represents the workload of the day/night task of s_i.

Next, according to equation (5), we estimate the number of devices required for each task of s_i (lines 9-10):

Input: The set of available devices **available_D**; the workload of day and night tasks WD_i and WN_i
Output: The estimated number of devices for each task number_DT_i and number_NT_i
(1) *total_AC* ← compute the total capabilities of all available devices;
(2) *average_AC* ← compute the average capability of all available devices
(3) **if** *total_AC* < $WD_i + WN_i$ **then**
(4) totalac_s_i = total_AC
(5) **else**
(6) totalac_s_i = WD_i+WN_i
(7) ac_DT_i = *total_AC* × (WD_i/WD_i+WN_i);
(8) ac_NT_i = *total_AC* × (WN_i/WD_i+WN_i);
(9) number_DT_i = round_off (ac_DT_i/average_AP);
(10) number_NT_i = round_off ((ac_NT_i/average_AP));
(11) **Return** [number_DT_i, number_NT_i];

Algorithm 3: estimate_number (*available_D*, WD_i, WN_i).

$$\begin{cases} \text{number\_DT}_i = \text{round\_off}\left(\dfrac{\text{ac\_DT}_i}{\text{average\_AC}}\right), \\ \text{number\_NT}_i = \text{round\_off}\left(\dfrac{\text{ac\_NT}_i}{\text{average\_AC}}\right), \end{cases} \tag{5}$$

where number_DT_i / number_NT_i is the estimated quantity of available devices required for the day and night tasks, *average_AC* is the average capability of all available devices, and *round_off* is the function that rounds off the estimated quantity of devices. For example, round_off (4.5) = 5 and round_off (4.4) = 4.

Subsequently, we use an example to illustrate how *estimate_quantity*() estimates the number of devices for each type of task of service.

Example 4. Combined with Example 2, the capability of available devices, from d_1 to d_5, is 1.1, 1.21, 1.35, 1.25, and 1.26, respectively. Supposing that the workload for the day and night tasks of s_i is 5 and 3, we compute the total capability of all available devices. That is, total_AC = 1.1 + 1.21 + 1.35+1.25 + 1.26 = 6.17. Then, the average capability of these available average_AC = 6.17/5 = 1.234. According to equation (3), totalac_s_i = 6.17, since total_AC < $WD_i + WN_i$ (6.17 < 5 + 3). Thus, we compute ac_DT_i = 6.17 × 5/(5 + 3) = 3.86 and ac_NT_i = 6.17 × 3/(5 + 3) = 2.31 as equation (4). Then, according to equation (5), we round off the estimated quantity of devices for each type of task of s_i, the estimated quantity of DT_i and NT_i is number_DT_i = round_off (3.86/1.234) = 3 and number_NT_i = round_off (2.31/1.234) = 2.

3.3. Searching for a Feasible Schedule. Since the quantity of devices is estimated by the average capability *average_AC* of all available devices, procedure *search_schedule*() should ensure that the average capability of the actual assigned devices is as close to *average_AC* as possible. The more closer to *average_AC* the average capability of the actual assigned device is, the smaller the values of |1 − Ave_Coverage| and *Coverage_Fairness* are. To achieve this goal, we propose a *PA* strategy to search for a feasible schedule. Besides, we define a *CAM* as an index to further its efficiency. Next, we specify the main idea of PA and *CAM* with the following example, combined with the procedure *search_schedule*(), as shown in Algorithm 4.

First, the capabilities of available device are sorted in ascending order (line 1). Then, as Figure 1 shows, it is divided into two subsets (i.e., AC_1 and AC_2) by the value of *average_AC* (line 2). Thus, AC_1 = {1.1, 1.21}, AC_2 = {1.25, 1.26, 1.25}. Next, we adopt PA strategy. Specifically, we take the first capability in AC_1 and AC_2 (lines 4-5), denoted by $\{L1, R1\} = \{1.1, 1.25\}$ (line 6). Thus, the corresponding device group $dp_1 = \{d_1, d_4\}$ (line 7). The average capability of dp_1 average_dp_1 (= (1.1 + 1.25/2) = 1.175) is treated as a trigger to find the next device group cp_2. To enable the average capability of dp_1 and dp_2 to maximally close to *average_AC*, the expected average capability of dp_2 Ep_2 = 2 × average_*AC* − average_dp_1 = 2 × 1.234 − 1.175 = 1.293 (line 8). To quickly pick up such dp_2, we precompute the *CAM* index as Figure 2 shows (line 3), which is defined as a square matrix of available_D. The PA strategy looks up the value that is closest to Ep_2 (=1.293) in this *CAM* index. Note that, to follow the hard constraint $Hard_1$ (i.e., each device is assigned to at most one task per service), dp_2 is selected from at the available device except d_1 and d_4 (lines 9, 11) as Figure 3 shows. Hence, $dp_2 = \{d_3, d_5\}$, because the average capability of d_3 and d_5 is 1.305, which is closest to 1.293 (line 10). Subsequently, the PA strategy computes the expected average capability of dp_i ($i \geq 3$) and locates dp_i in the same way until the total quantity of devices in the selected groups equals the estimated quantity (line 12). It is worth noting that if the estimated quantity of devices is odd, the last device is regarded as a group. Selecting the last device also follows the principle that the average capability of the group (i.e., the capability of the last device) is maximally close to the expected average capability. Thus, we generate a candidate assignment of device combinations (DC_DT_i and DC_NT_i), where the devices d_1, d_4, d_5 are assigned to DT_i and d_2, d_3 are assigned to NT_i (lines 13–16). Subsequently, we select the first capability in AC_1 and the second one in AC_2, denoted by $\{L1, R2\}$ = {1.1, 1.26}, and the corresponding device group

Input: The capabilities of available devices **AC**; The estimated device quantity for each task number_DT$_i$, number_NT$_i$
Output: The feasible schedule **Schedule**
(1) AC←sort AC in ascending order;
(2) AC$_1$, AC$_2$←average_AC;
(3) *CAM* ← compute the average value of any two capabilities in AC
(4) **for** $c_j \in$ AC$_1$ **do**
(5) **for** $c_k \in$ AC$_2$ **do**
(6) $\{L1, R1\} \leftarrow (c_j, c_k)$;
(7) dp_1 find the corresponding devices of c_j, c_k in AC
(8) Ep$_2$2 × *average*_AP − average_dp$_1$;
(9) **Remove** $dp_1 \in CAM$, AC$_1$ and AC$_2$;
(10) $dp_2 \leftarrow$ search in *CAM*;
(11) **Remove** $dp_2 \in CAM$, AC_1 and AC_2;
(12) search dp_i until device quantity of $\{dp_1, dp_2, \ldots, dp_{i-1}\}$ equals to number_DT$_i$
(13) $DC\_DT_i \leftarrow \{dp_1, dp_2, \ldots, dp_{i-1}\}$;
(14) **Replace** number_DT$_i$ with number_NT$_i$;
(15) DC_NT$_i$ repeat the search operations of DT_i
(16) **add** (DC_DT$_i$, DC_NT$_i$) into assignment
(17) the feasible schedule *fs* ← TOPSIS(assignment)
(18) **Return** *fs*

Algorithm 4: search_schedule (available_D, totalc_DT$_i$, totalc_NT$_i$, number_DT$_i$, number_NT$_i$).

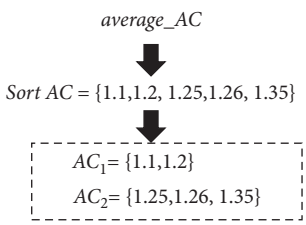

Figure 1: Dividing the capabilities set into two subsets.

		d_1	d_2	d_3	d_4	d_5
		ac_1	ac_2	ac_3	ac_4	ac_5
d_1	ac_1	1.1	1.155	1.225	1.175	1.18
d_2	ac_2	1.155	1.21	1.28	1.23	1.235
d_3	ac_3	1.225	1.28	1.25	1.3	1.305
d_4	ac_4	1.175	1.23	1.3	1.25	1.255
d_5	ac_5	1.18	1.235	1.305	1.255	1.26

Figure 2: The example of establishing capability average matrix.

		d_1	d_2	d_3	d_4	d_5
		ac_1	ac_2	ac_3	ac_4	ac_5
d_1	ac_1	1.1	1.155	1.225	1.175	1.18
d_2	ac_2	1.155	1.21	1.28	1.23	1.235
d_3	ac_3	1.225	1.28	1.25	1.3	1.305
d_4	ac_4	1.175	1.23	1.3	1.25	1.255
d_5	ac_5	1.18	1.235	1.305	1.255	1.26

Figure 3: The example of searching dp_2.

$dp_1 = \{d_1, d_5\}$ is utilized to generate the next candidate assignment until all combinations of capability between AC_1 and AC_2 are listed.

These candidate assignments will be evaluated by TOPSIS [19] to select the feasible schedule (line 17). The reason why we use TOPSIS is that it is a multicriteria decision analysis method [20] and is widely used in evaluating a result with different dimensions. TOPSIS orthogonalizes optimization goals; that is, all optimization goals have the same direction of optimization. Next, TOPSIS standardizes all optimization goals since the optimization goals have different magnitudes. Finally, TOPSIS evaluates all candidate assignments and generates their scores; the higher the score is, the higher the quality of this assignment is. The detailed steps of TOPSIS will be described in the experiment section.

4. Discussion

We discuss the space and time complexities below.

4.1. Space Complexity Comparison. In the worst case, the space of our approach is $O(n \cdot (m^2/2))$, where n denotes the number of services in the scheduling horizon S and m represents the number of devices. As for the method of enumerating all possible schedules, its space is $O(n \cdot (\sum_{t=0,1,\ldots,m} 2^t \cdot C_m^t))$, where n denotes the number of services in the scheduling horizon S, t is the number of devices whose capabilities maximally minimize |1 − Ave_Coverage| and *Fairness_Coverage*, and C_m^t represents the number of combinations of selecting t devices from the devices set whose size is m. Hence, the space of our approach is square, and that of traversing is exponential.

$$\begin{aligned}&\lim_{n\longrightarrow\infty} n\times\frac{m^2}{2}/n\times\left(\sum_{t=0,1,\ldots,m}2^t\cdot C_m^t\right)\\&\leq\lim_{m\longrightarrow\infty}\frac{m^2}{\sum_{t=0,1,\ldots,m}C_m^t}\\&=\lim_{m\longrightarrow\infty}\frac{m^2}{2^m}=0.\end{aligned}\tag{6}$$

4.2. Time Complexity. Our approach generates a feasible assignment by the *CAM* index, which is composed of m^2 average capabilities. Hence, a feasible assignment requires $O(m^2)$ time and all feasible assignments of n services need $O(n{\cdot}m^4)$ time. As for enumerating all possible schedules, it requires $O(n\cdot(\sum_{t=0,1,\ldots,m}2^t\cdot C_m^t))$ time. This time complexity is more than $O(2^m)$, and $O(2^m){>}r$ bin $O(\mathrm{m}^4)$.

5. Experiments

In this section, we experimentally evaluate the efficiency and effectiveness of our proposed solution *PACAM* against the state-of-the-art methods. We implement our algorithm in Python and adopt the Python implementation of all competitors based on the following methods: (1) LP [21], (2) IP [22], (3) MIP [16], (4) IPSO [23], and (5) MOEAD [24]. The solver used in this article is Gurobi solver 9.1 [25]. It is worth noting that since all the above-mentioned approaches are universal methods, they can not be used directly; we make minor modifications to adapt them to our studied problem. In addition, to compare the performance of *PACAM* with the other five methods, we (1) report the response time of each method generating the same feasible schedule results and (2) report the TOPSIS score of each method under the same response time, in order to have a fairer and more accurate comparison.

Besides, all evaluations in this section are based on a mixture of real and synthetic datasets, which are presented in Table 5. The real part comes from the China Telecom company, containing the tasks of six months from July 2020 to December 2020. In Table 5, the number of services of each dataset is listed; it is worth noting that the tasks of each service s_i in each dataset are divided into two parts, treated as the workloads WD_i and WN_i for the day and night tasks of s_i, respectively. The synthetic part is the devices, which are synthesized from the real devices of China Telecom company, as shown in Table 5. Specifically, each month contains five synthetic device sets, whose quantities are 20, 40, 60, 80, and 100. Each device in these device sets has a capability. Note that each experiment will randomly choose the corresponding quantity of devices for ten times to be executed, and the average measurement is reported. All the experiments are conducted on a server machine with an Intel(R) Xeon(R) CPU E5-2637 3.50 GHz processor and 8 GB RAM, running Windows 10 with Python 3.8.

5.1. Metrics. Each potential schedule will be evaluated for facilitating generating the feasible schedule, and we adopt TOPSIS [19] to achieve this step. The main step of TOPSIS is presented as follows.

First, the optimization goals need to be transformed into such an indicator, whose value is larger, resulting in a feasible with better quality, as TOPSIS asks. Hence, we transform *Ave_Coverage* according to the following equation:

$$\begin{aligned}\overline{x_i}&=1-\frac{\left|x_i-x_{\mathrm{best}}\right|}{M},\\\text{s.t. M}&=\max\left\{\left|x_i-x_{\mathrm{best}}\right|\right\},\end{aligned}\tag{7}$$

where x_i denotes the value of *Ave_Coverage* for the i^{th} schedule and x_{best} is the best value of *Ave_Coverage* for all schedules.

Coverage_Fairness is transformed based on the following equation:

$$\overline{y_i}=\frac{1}{y_i},\tag{8}$$

where y_i denotes the value of *Coverage_Fairness* for i^{th} schedule.

Next, we normalize these transformed optimization goals as follows:

$$\mathrm{normalized\_value}_{ij}=\frac{\mathrm{value}_{ij}}{\sqrt{\sum_{i=1}^{n}value_{ij}^2}},\tag{9}$$

where value_{ij} is the value of i^{th} optimization goal for j^{th} potential schedule and n denotes the number of potential schedules.

Then, we compute the distance between the optimal/worst schedule and each potential schedule, according to the following:

$$\begin{cases}\mathrm{distance\_opt}=\sqrt{\sum_{j=1}^{2}\left(\mathrm{opt}-\mathrm{normalized\_value}_{ij}\right)^2},\\\mathrm{distance\_wor}=\sqrt{\sum_{j=1}^{2}\left(\mathrm{wor}-\mathrm{normalized\_value}_{ij}\right)^2},\end{cases}\tag{10}$$

where *opt/wor* denotes the values of optimization goals for the optimal/worst schedule among all potential schedules.

Finally, we score each potential schedule according to the following equation and select the feasible schedule:

$$\mathrm{score}_i=\frac{\mathrm{distance\_wor}}{\mathrm{distance\_opt}+\mathrm{distance\_wor}}.\tag{11}$$

5.2. Experiment Setting. We perform nine sets of experiments to evaluate the performance of PACAM, where EXP1 to EXP6 are used to evaluate the overall performance among PACAM and five alternatives and EXP7 to EXP9 present the

Table 5: The datasets used in experiments.

Datasets	Services	Tasks	Device quantity
July 2020	30	37,691	81
Aug. 2020	31	38,037	60
Sept. 2020	31	35,991	76
Oct. 2020	30	37,110	70
Nov. 2020	31	38,133	86
Dec. 2020	30	36,510	79

internal performance of PACAM. The parameters in each experiment are illustrated in Table 6.

EXP1 and EXP2 evaluate the impact of varying the datasets; we fix the number of devices to 100 and set the maximal consecutive services k to 4 services. Note that EXP1 generates the schedule with the same quality on all six datasets to report the response time, and EXP2 runs the same response time on all six datasets to report the quality of the generated schedule. EXP3 and EXP4 evaluate the impact of varying k; we fix the number of devices to 100 and run EXP3 on November to report the response time for generating the schedule with the same quality and run EXP4 on November to report the quality of generated schedule with the same running time. EXP5 and EXP6 evaluate the impact of varying the number of devices; we set the maximal consecutive services k to 4 and run EXP5 on November to report the response time for generating the schedule with the same quality and run EXP6 to report the quality of generated schedule with the same running time.

All internal experiments (i.e., from EXP7 to EXP9) are performed, considering generating the schedule with the same quality as the end signal. Besides, EXP7 evaluates the internal impact of varying the datasets for PACAM; we fix the number of devices to 100, set the maximal consecutive services to 4, and report the response time. EXP8 evaluates the internal impact of varying k; we fix the number of devices to 100 and run it on November to report response time. EXP9 evaluates the internal impact of varying the number of devices; we set the maximal consecutive services to 4 and run it on November to report the response time.

5.3. Overall Performance

5.3.1. EXP 1: Search Efficiency. The first set of experiments verifies the performance of PACAM by varying datasets compared with the other five alternative methods. The result is illustrated in Figure 4(a). The first observation is that PACAM is the best in all cases, with IP, LP, and MIP in the second place, they are comparable, and then IPSO and MOEAD are the worst. Particularly, on all six datasets, PACAM outperforms IP, LP, and MIP by two orders of magnitude, and it is faster than IPSO and MOEAD by 316.23 and 2511.89 times at least, respectively. This is because that IP, LP, and MIP need to consider all the potential schedules and MOEAD and IPSO need a large number of iterations to generate a feasible schedule due to the random nature of searching strategies, while PACAM preestimates the number of devices required for each type of task, which effectively shrinks the number of potential schedules. The second observation is that PACAM is more stable than MOEAD because PACAM searches for the feasible schedule within a small search range, owing to its preestimating strategy. While MOEAD generates the optimal schedule by mutation and crossover operations, these operations contain a random mechanism, which results in the instability of newly generated solutions.

Table 6: The parameters used in experiments.

EXPs	Data sets	Number of devices	k	Same run time	Same quality
EXP1	—	100	4		✓
EXP2	—	100	4	✓	
EXP3	Nov.	100	—		✓
EXP4	Nov.	100	—	✓	
EXP5	Nov.	—	4		✓
EXP6	Nov.	—	4	✓	
EXP7	—	100	4		✓
EXP8	Nov.	100	—		✓
EXP7	Nov.	—	4		✓

5.3.2. EXP 2: Search Effectiveness on Running the Same Time. Compared with EXP 1, EXP 2 runs under the condition of running the same time and reports the TOPSIS score of each method as illustrated in Figure 4(b). It is seen that when varying datasets, the TOPSIS score of PACAM changes slightly, and it is the highest. The reason is that PACAM adopts the pairwise-allocated strategy, which assigns devices to the tasks following the principle that the total capabilities of devices assigned to one task should be maximally close to the workload of this task. This guarantees that the average coverage will be maximally close to 1 and the coverage fairness will be reduced to the minimum. Hence, the corresponding TOPSIS will perform best. Three mathematical methods consider a prohibitive number of potential schedules, and thus they need more time to generate the feasible schedule. IPSO searches for the feasible schedule according to one previous solution nearest to the fixed schedule, which easily leads to locally optimal solutions. As for MOEAD, the limited running time does not allow MOEAD to perform enough generation operations, which influences the quality of the generated schedule.

5.3.3. EXP 3: Effect of k on Search Efficiency. The third set of experiments aims to explore the impact of varying maximal consecutive services k on search performance. The result is shown in Figure 5(a). The first observation is that the running time of all methods, except MOEAD, changes slightly with the growth of k. This is because PACAM uses the preestimating strategy so that varying the maximal consecutive services has nearly no effect on its efficiency. IP, LP, and MIP search for the feasible schedule in a huge range; the internal algorithms in solvers such as the branch and bound method help generate stable solutions. IPSO falls into a local optimum since its search strategy is based on the previous solution nearest to the fixed schedule. On the other hand, MOEAD generates a feasible schedule based on the

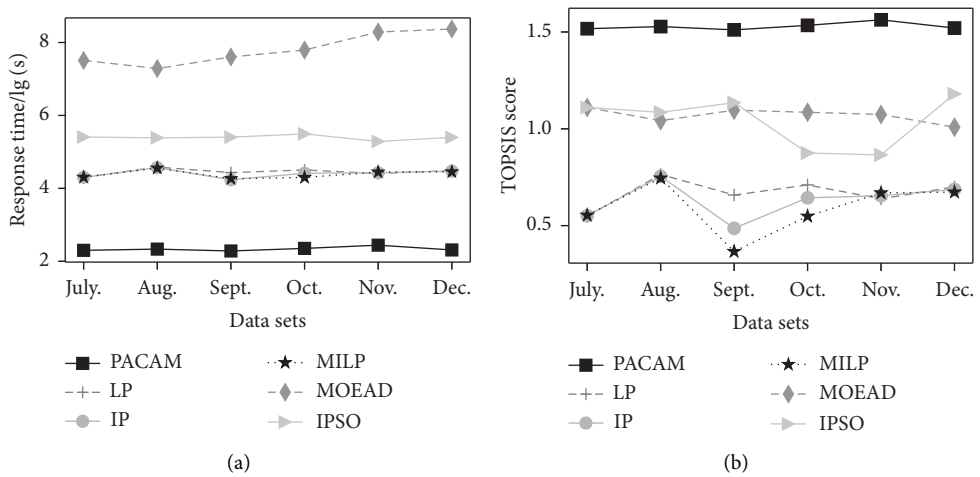

Figure 4: Overall effectiveness and efficiency with different datasets. (a) Fixing schedule quality. (b) Fixing response time.

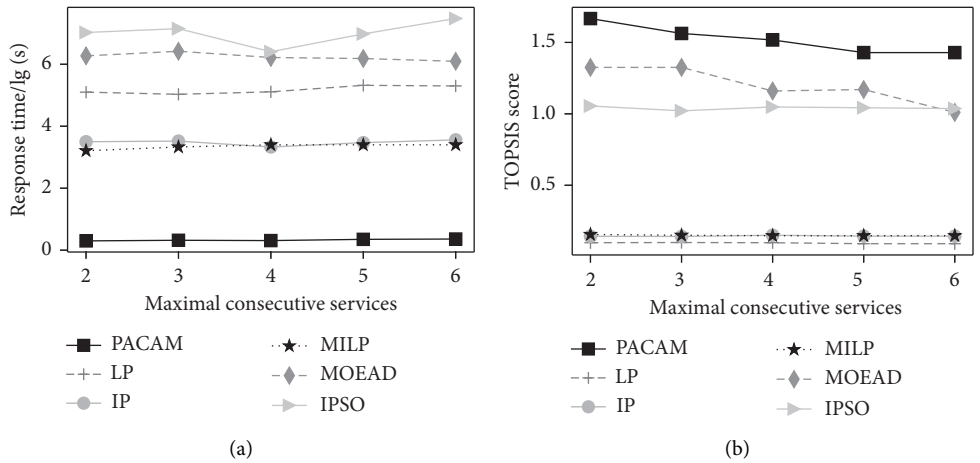

Figure 5: Overall effectiveness and efficiency with varying k. (a) Fixing schedule quality. (b) Fixing response time.

random nature of the search strategy, and its response time is dependent on this rather than k. Thus, the response time of MOEAD fluctuates greatly. The second observation is that PACAM has the lowest elapsed time. The reason lies in two aspects. First, while searching for the feasible schedule, PACAM preprunes a large number of potential schedules and finds the feasible schedule based on pairwise-allocated strategy. Second, CAM is established to boost efficiency.

5.3.4. EXP 4: Effect of k on Search Effectiveness with Same Running Time. EXP 4 runs under the same condition as EXP3 and reports the TOPSIS score of each method. The result is plotted in Figure 5(b). The first observation is that the TOPSIS score of PACAM increases as k ascends. The reason is that a larger k means fewer rest services in a scheduling horizon and more available devices to be assigned in a day, leading to the total capabilities of devices assigned to one task being closer to its workload, and hence a higher TOPSIS score. The second observation is that the TOPSIS score of PACAM is still the highest under the different k. It is because the pairwise-allocated strategy in PACAM follows the principle that the selected device group should be the one making the average capability of all selected device groups nearest to that of all devices. Thus, the total capabilities of all assigned devices are maximally equal to the workload of the corresponding tasks.

5.3.5. EXP 5: Effect of the Number of Devices on Search Efficiency. The fifth set of experiments evaluates the impact of the number of devices on search efficiency. The result is depicted in Figure 6(a). The first observation is that the response time of PACAM, IP, LP, and MIP increases as the number of devices grows. The reason is that, for PACAM, an increasing number of devices means more potential device groups, which requires PACAM to spend more time searching for the suitable device group. Three mathematical methods require more response time since the number of potential device group combinations increases with the number of devices grows. The second observation is that the running time of MOEAD and IPSO fluctuates with the

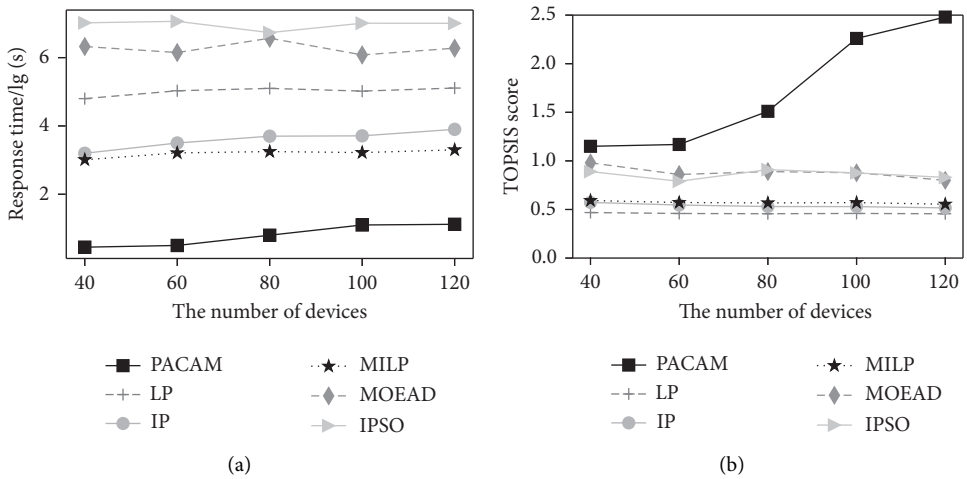

Figure 6: Overall effectiveness and efficiency with the number of devices. (a) Fixing schedule quality. (b) Fixing response time.

increase in the number of devices. The reason is that to search for the feasible schedule, MOEAD and IPSO need a large number of generation and iteration operations, where there is a great deal of randomness.

5.3.6. EXP 6: Effect of the Number of Devices on Search Effectiveness. Figure 6(b) shows the result of each method by varying the number of devices. It is observed that the TOPSIS score of PACAM increases as the number of devices grows. It is because that more device groups are available with an increasing number of devices, which means there is a higher possibility for PACAM selecting the device groups whose capability is nearest to the workload of tasks. As for the others, under limited running time, their mechanisms are time-consuming, and the generated feasible schedules are of low quality. Again, the TOPSIS score of PACAM is the highest in all cases.

5.4. Internal Performance

5.4.1. EXP 7: Internal Performance vs. Different Datasets. The seventh set of experiments evaluates the internal impact of the performance of the *PA* strategy and *CAM* by varying the datasets. We compare PACAM with two alternative methods, i.e., PACAM-NoCAM and *Traverse*, respectively. PACAM-NoCAM removes the *CAM*, and *Traverse* evaluates all potential schedules. The result is illustrated in Figure 7(a). It is observed that PACAM is faster than *PACAM*-NoCAM and *Traverse* on all datasets. In particular, PACAM is 380 times faster than PACAM-NoCAM and outperforms *Traverse* by two orders of magnitude on average, respectively. This is because, compared to *Traverse*, PACAM and PACAM-NoCAM contain PA, which greatly reduces the number of potential schedules. This indicates that PA effectively shrinks search range and improves efficiency. In addition, PACAM adopts *CAM* to boost efficiency, and based on *CAM*, PACAM outperforms PACAM-NoCAM by 4.47 times; this finding indicates that *CAM* further improves the efficiency of search.

5.4.2. EXP 8: Internal Performance vs. k. The eighth set of experiments verifies the internal performance for PACAM by varying k. The result is illustrated in Figure 7(b). The first observation is that the response time of all methods remains stable as the number of consecutive services grows. It is because both PACAM and PACAM-NoCAM adopt the pairwise-allocated strategy, shrink the number of potential schedules, and search for the feasible schedule in a small search range, which provides the stability of PACAM and PACAM-NoCAM. As for *Traverse*, once it finds the schedule, the response time is reported. Hence, the response time of *Traverse* remains stable. The second observation is that PACAM is the best in all cases, with PACAM-NoCAM in the second place, and then *Traverse* is the worst. The reason is that the number of potential schedules that are evaluated by *Traverse* is large; hence, *Traverse* requires a large amount of time consumption. PACAM-NoCAM adopts the PA strategy to reduce the number of potential schedules and searches for the feasible schedule in a small range, which effectively reduces the time cost. In addition, based on the PA strategy, PACAM utilizes the CAM to improve efficiency further.

5.4.3. EXP 9: Internal Performance vs. the Number of Devices. The ninth set of experiments explores the effect of the number of devices on the internal performance of PACAM. The result is plotted in Figure 7(c). The first observation is that the response time of traversing is exponential because the number of potential schedules grows exponentially as the number of devices increases. The second observation is that the response time of PACAM and PACAM-NoCAM keeps stable since estimating the number of devices prunes plenty of unfeasible schedules, leading to less time in selecting suitable device groups. The third observation is that compared with PACAM-NoCAM, the response time of PACAM is much shorter. The reason is that PACAM uses CAM for fast finding the suitable device groups.

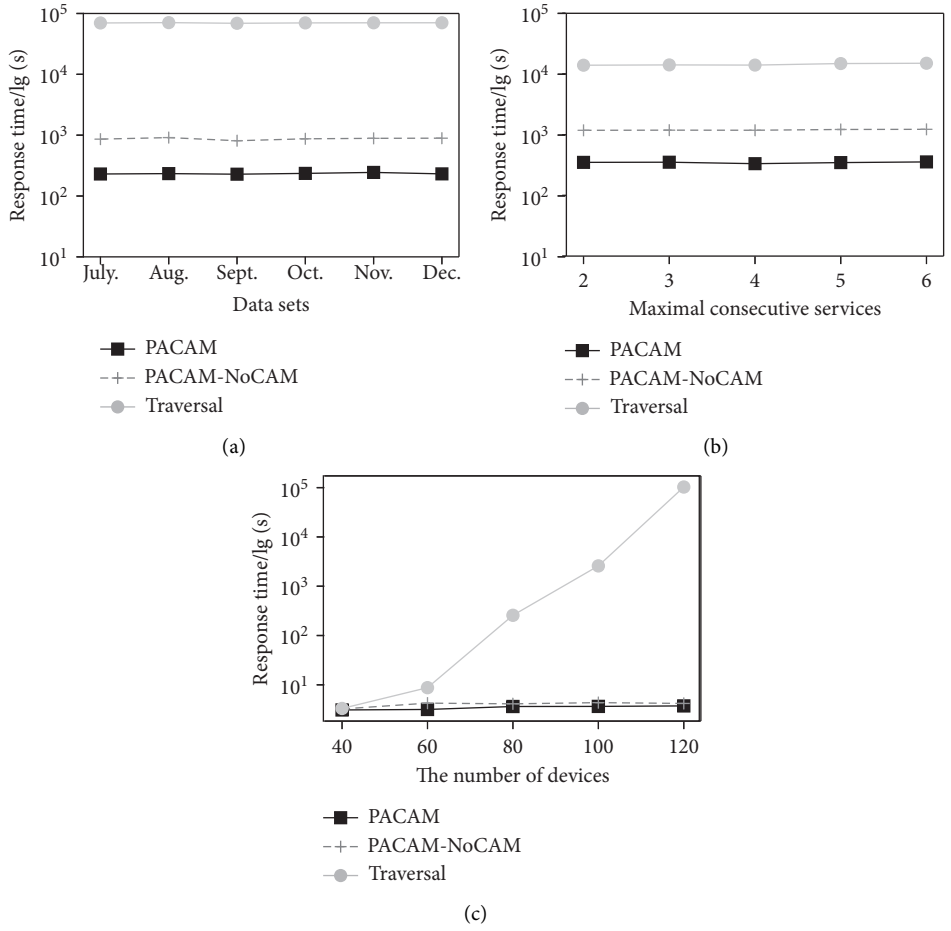

Figure 7: Internal performance of PACAM by varying parameters. (a) Response time vs. different data sets. (b) Response time vs. k. (c) Response time vs. the number of devices.

6. Related Work

Currently, algorithms to address task scheduling problems of edge computing are divided into three categories. The first category is to adopt the method of mathematical programming. Regularly, it models the task scheduling problem by complex mathematical models, then computes the feasible schedules by operating a series of solvers like MIP solver [13–17] on this problem model. Although this category of the method has high effectiveness, a large amount of computation leads to low efficiency and high response time. These methods do not provide the allocated strategy and search strategy as PACAM does and limi themselves to similar trips or other mathematical methods.

The second category is the approximate algorithm. Many studies are adopting approximate algorithms in such resource scheduling. Badri et al. [26] modeled the application placement of the MEC system and adopted the parallel sample averaging approximation (SAA) algorithm to solve it. Guo et al. [27] treated the edge-cloud placement problem as a multiobjective optimization problem and addressed it by k-means and hybrid quadratic programming. Fang [28] described a multiuser resource allocation problem in edge computing and proposed an approximate algorithm for local search to solve this problem. The basic idea of these algorithms is to model their problem and adopt the existing approximate methods for generating feasible solutions. In addition, some works assigned tasks to devices with proper skill type by collaborative filtering mechanism [29], but it is out of our consideration. In summary, these methods have shortages, such that they are easy to fall into a local optimum and the performance of the solutions can not be guaranteed due to randomness. Moreover, in our work, we propose PACAM, which strategically assigns devices and defines a matrix-based index to speed up searching for the feasible schedule.

The third category is the heuristic algorithm. Nowadays, an increasing number of ways to solve task scheduling problems in edge computing is heuristic algorithms. Considering some phenomenons in nature, heuristic algorithms abstracted this similarity into their methods to address the task scheduling problem [30]. Hong et al. [24] modeled their resource scheduling problem as a multiobjective problem and adopted the multiobjective evolutionary algorithm based on decomposition (MOEAD) to generate feasible solutions. In addition, some works [31, 32] used the nondominated sorting genetic algorithm (NSGA) to deal with the task scheduling problem containing multiple objectives.

Besides, the authors in [23] proposed a PSO-based heuristic strategy to solve the joint problem of service placement and task provisioning. This algorithm solved the resource scheduling problem by greedy-based and genetic-based algorithms. However, they are easy to fall into a local optimal solution. Besides, most of them require too much domain knowledge to adjust the parameters, and it is impossible to get feasible parameters efficiently. In our work, PACAM preestimates the number of devices required for each type of task, which avoids wide-range search. PACAM adopts a series of strategies to shrink the number of potential schedules, such as the PA strategy and CAM and effectively assign devices to the corresponding tasks of each day.

7. Conclusion

This article proposes PACAM, a PA strategy and CAM-based approach, to solving the task scheduling problem. Devices are assigned to tasks of each service in the scheduling horizon; meanwhile, the hard constraints must be followed and the soft constraint should be maximally satisfied. Based on these, PACAM estimates the number of devices required for each type of task of each service in the scheduling horizon. Then, PACAM adopts the PA strategy to assign the corresponding number of devices to tasks. To facilitate finding the device groups we need, PACAM takes CAM as an index to speed up this process. Thus, we just spent a few computations to find the final feasible schedule. Extensive experimental evaluation demonstrates the effectiveness and efficiency of our proposed approach. In the future, we intend to explore how to assign large-scale devices to tasks with more optimization goals.

Data Availability

The data will be made available on reasonable request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National R&D Program of China, under Grant no. 2018YFB1402800, and the Fundamental Research Funds for the Provincial Universities of Zhejiang under Grant no. RF-A2020007.

References

[1] Z. Ning, K. Zhang, X. Wang et al., "Intelligent edge computing in internet of vehicles: a joint computation offloading and caching solution," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2212–2225, 2020.

[2] J. Lin, W. Yu, X. Yang, P. Zhao, H. Zhang, and W. Zhao, "An edge computing based public vehicle system for smart transportation," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 11, pp. 12635–12651, 2020.

[3] H. Wang, J. Gong, Y. Zhuang, H. Shen, and J. Lach, "Healthedge: task scheduling for edge computing with health emergency and human behavior consideration in smart homes," in *Proceedings of the 2017 IEEE International Conference on Big Data (Big Data)*, pp. 1213–1222, IEEE, Shenzhen, China, August 2017.

[4] H. Liu, S. Li, and W. Sun, "Resource allocation for edge computing without using cloud center in smart home environment: a pricing approach," *Sensors*, vol. 20, no. 22, p. 6545, 2020.

[5] H. Gao, X. Qin, R. J. D. Barroso, W. Hussain, Y. Xu, and Y. Yin, "Collaborative learning-based industrial iot api recommendation for software-defined devices: The implicit knowledge discovery perspective," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2020.

[6] H. Gao, K. Xu, M. Cao, J. Xiao, Q. Xu, and Y. Yin, "The deep features and attention mechanism-based method to dish healthcare under social iot systems: An empirical study with a hand-deep local-global net," *IEEE Transactions on Computational Social Systems*, 2021.

[7] Y. Yin, Z. Cao, Y. Xu, H. Gao, R. Li, and Z. Mai, "Qos prediction for service recommendation with features learning in mobile edge computing environment," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 4, pp. 1136–1145, 2020.

[8] Y. Li and S. Wang, "An energy-aware edge server placement algorithm in Mobile Edge Computing," in *Proceedings of the 2018 IEEE International Conference on Edge Computing (EDGE)*, San Francisco, CA, USA, July 2018.

[9] J. Meng, C. Zeng, H. Tan, Z. Li, B. Li, and X. Y. Li, "Joint heterogeneous server placement and application configuration in edge computing," in *Proceedings of the 2019 IEEE 25th International Conference on Parallel and Distributed Systems*, Tianjin, China, January 2020.

[10] K. Xiao, Z. Gao, W. Qian, and Y. Yang, "A Heuristic Algorithm Based on Resource Requirements Forecasting for Server Placement in Edge Computing," in *Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing (SEC)*, Seattle, WA, USA, October 2018.

[11] J. Zhou and X. Zhang, "Fairness-aware task offloading and resource allocation in cooperative mobile edge computing," *IEEE Internet of Things Journal*, 2021.

[12] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *Proceedings of the 2019 IEEE International Conference on Computer Communications (IEEE INFOCOM 2019)*, Paris, France, May 2019.

[13] S. Yang, F. Li, M. Shen, X. Chen, X. Fu, and Y. Wang, "Cloudlet placement and task allocation in mobile edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5853–5863, 2019.

[14] M. Breitbach, D. Schäfer, J. Edinger, and C. Becker, "Context-aware data and task placement in edge computing environments," in *Proceedings of the 2019 IEEE International Conference On Pervasive Computing And Communications (PerCom. IEEE)*, pp. 1–10, Kyoto, Japan, March 2019.

[15] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of experience (qoe)-aware placement of applications in fog computing environments," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 190–203, 2019.

[16] R. Mahmuda, S. N. Sriramab, K. Ramamohanaraoa, and R. Buyya, "Profit-aware application placement for integrated fog–cloud computing environments," *Journal of Parallel and Distributed Computing*, vol. 135, pp. 177–190, 2020.

[17] S. Hu and G. Li, "Dynamic request scheduling optimization in mobile edge computing for iot applications," *IEEE Internet of Things Journal*, vol. 7, no. 2, pp. 1426–1437, 2019.

[18] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, "A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 904–916, 2019.

[19] G.-H. Tzeng and J.-J. Huang, *Multiple Attribute Decision Making: Methods and Applications*, CRC Press, Boca Raton, FL, USA, 2011.

[20] V. Balioti, C. Tzimopoulos, and C. Evangelides, "Multi-criteria decision making using TOPSIS method under fuzzy environment. Application in spillway selection," *Proceedings*, vol. 2, no. 11, 2018.

[21] P. Strandmark, Y. Qu, and T. Curtois, "First-order linear programming in a column generation-based heuristic approach to the nurse rostering problem," *Computers & Operations Research*, vol. 120, Article ID 104945, 2020.

[22] R. Bürgy, H. Michon-Lacaze, and G. Desaulniers, "Employee scheduling with short demand perturbations and extensible shifts," *Omega*, vol. 89, pp. 177–192, 2019.

[23] A. Mseddi, W. Jaafar, H. Elbiaze, and W. Ajib, "Joint container placement and task provisioning in dynamic fog computing," *IEEE Internet of Things Journal*, vol. 6, no. (6), pp. 10028–10040, 2019.

[24] F. Hong, H. Chen, B. Cao, and J. Fan, "A moead-based approach to solving the staff scheduling problem," in *Proceedings of the International Conference On Collaborative Computing: Networking, Applications And Worksharing*, pp. 112–131, Springer, Shanghai, China, October 2020.

[25] I. Gurobi Optimization, *Gurobi Optimizer Reference Manual*, Gurobi Optimization LLC, Houston, TX, USA, 2018.

[26] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Energy-aware application placement in mobile edge computing: A stochastic optimization approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, 2020.

[27] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, and C. H. Hsu, "User allocation-aware edge cloud placement in mobile edge computing," *Software: Practice and Experience*, vol. 50, no. 5, pp. 489–502, 2020.

[28] S. L Fang, "Cost-efficient resource provision for multiple mobile users in fog computing," in *Proceedings of the 2019 IEEE 25th International Conference on Parallel and Distributed Systems*, December 2019.

[29] X. Yang, S. Zhou, and M. Cao, "An approach to alleviate the sparsity problem of hybrid collaborative filtering based recommendations: the product-attribute perspective from user reviews," *Mobile Networks and Applications*, vol. 25, no. 2, 2020.

[30] Q. Luo, C. Li, T. H. Luan, and Y. Wen, "Optimal utility of vehicles in lte-v scenario: An immune clone-based spectrum allocation approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, pp. 1–12, 2018.

[31] K. Peng, M. Zhu, Y. Zhang et al., "An energy-and cost-aware computation offloading method for workflow applications in mobile edge computing," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, pp. 1–15, 2019.

[32] X. Xu, H. Cao, Q. Geng, X. Liu, and C. Wang, "Dynamic Resource Provisioning for Workflow Scheduling under Uncertainty in Edge Computing Environment," *Concurrency and Computation Practice and Experience*, 2020.