Task Scheduling with Multicore Edge Computing in Dense Small Cell Networks

Wei Kuang Lai ¹, Senior Member, IEEE, Chin-Shiuh Shieh ², Member, IEEE, and Yen-Ping Chen ¹

¹ Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan, ROC
² Department of Electronic Engineering, National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan, ROC

Corresponding author: Chin-Shiuh Shieh (e-mail: csshieh@nkust.edu.tw).

This work was supported in part by the Minister of Science and Technology, Taiwan under Grant 109-2221-E-110-049-MY2 and MOST 110-2221-E-992-012.

ABSTRACT As a reaction and complement to cloud computing, edge computing is a computing paradigm designed for low-latency computing. Edge servers, deployed at the boundary of the Internet, bridge those distributed end devices and the centralized cloud server, forming a harmonic architecture with low latency and balanced loadings. Elaborated task scheduling, including task assignment and processor dispatching, is essential to the success of edge computing systems in dense small cell networks. Plenty of issues need to be considered, such as servers’ computing power, storage capacity, loadings, bandwidth and tasks’ sizes, delays, partitionability, etc. This study contributes to the task scheduling for multicore edge computing environments. We first show that this scheduling problem is an NP-hard problem. An efficient and effective heuristic is then proposed to tackle the problem. Our Multicore Task assignment for maximum Rewards (MAR) scheme differs from most previous schemes in jointly considering all three critical factors: namely task partitionability, multicore, and task properties. A task’s priority is decided by its cost function, which takes into account the task’s size, deadline, partitionability, cores’ loadings, processing power, and so forth. First, tasks from end devices are assigned to edge servers considering servers’ loadings and storage. Next, tasks are assigned to the cores of the selected server. Simulations compare the proposed scheme with First-Come-First-Serve (FCFS), Shortest Task First (STF), Delay Priority Scheduling (DPS), and Green Greedy Algorithm (GGA). Simulations demonstrate that the task completion ratio can be significantly increased, and the number of aborted tasks can be greatly reduced. Compared with FCFS (First-Come-First-Serve), STF (Shortest Task First), DPS (Delay Priority Scheduling), and GGA (Green Greedy Algorithm), the improvement in task completion ratio for hotspots is up to 26%, 25%, 22%, and 9%, respectively.

INDEX TERMS Task scheduling, edge computing, multicore.

I. INTRODUCTION

As applications evolve, computing paradigms shift towards three directions, centralized, distributed, and edge computing. Centralized computing, such as cloud computing [1], facilitates the management of computing resources and the processing of aggregated data. However, the unavailability of networking function, even temporarily, disables the entire system. On the other hand, distributing computing, such as the end devices of monitoring and control systems, is immune from the problem of a single point of failure. However, there is considerable latency in performing data aggregation and fusion.

In between the two extremes, the rise of the Internet of Things (IoT) [2] inspires the emerging of a new computing paradigm called edge computing [3]. In an edge computing environment, edge servers are deployed in the proximity of the access networks. Edge servers bridge those end devices at the network periphery and the cloud server in the core network. End devices are, in general, inferior in their computing power and storage capacity. End devices offload their computing tasks to edge servers rather than the cloud server as their first choice. Upon the completion of a task, edge servers return the results to the end device and pass the deputated information to the cloud server. A computing paradigm with prompt responses and balanced loading can be expected with edge computing.

The concept of edge computing is widely embraced by public and private branches. European Telecommunications
Standard Institute (ETSI) established related standards and regulations. Numerous industrial parties actively participate in the process, including Nokia, Intel, Vodafone, IBM, Huawei, etc. According to the white paper by ETSI, edge computing distinguishes itself with the following characteristics: on-premises, proximity, lower latency, location awareness, and network context information. The 5G Infrastructure Public-Private Partnership (5G PPP) also acknowledges that edge computing is an emerging key technology for mobile communication [4]. Edge computing with mobile communication is expected to be an important trend in the future [5].

A straightforward integration is to deploy the edge servers at the Radio Access Network (RAN), as illustrated in Figure 1, which is a typical scenario we intend in this study. In Figure 1, task assignment and server/processor dispatch strongly affect the performance of an edge computing system. Various issues need to be considered, such as servers’ computing power, storage, loadings, bandwidth and tasks’ sizes, delaines, partitionability, etc.

Our contributions to task scheduling in multicore edge computing environments are summarized bellows:

- We first show that the task assignment and server/processor dispatch problem is an NP-hard problem.
- The problem is approached with an efficient and effective two-step heuristic. First, tasks from end devices are assigned to edge servers factoring in servers’ loadings and storage. Next, the dispatched tasks onto the selected servers are assigned to the cores of the selected server.
- A task’s priority is decided by its cost function, which considers essential factors, including tasks’ size, deadlines, partitionability, cores’ loadings, and processing power.
- The proposed Multicore task Assignment scheme for maximum Rewards (MAR) scheme effectually improves the task completion ratio. When comparing the MAR scheme with FCFS (First-Come-First-Serve), STF (Shortest Task First), DPS (Delay Priority Scheduling), and GGA (Green Greedy Algorithm), simulation results show the improvement in task completion ratio for hotspots is up to 26%, 25%, 22%, and 9%, respectively.

The rest of this paper is organized as follows. Previous works are briefly reviewed in Section II, highlighting their relative merits and limitations. We define the system model and formulate the scheduling problem in Section III. Then, our MAR scheme is presented in Section IV with an in-depth explanation and discussion. A series of simulations demonstrating the feasibility and effectiveness of the proposed approach are presented in Section V. Finally, some conclusions are drawn in Section VI.

II. RELATED WORKS

End devices are generally equipped with limited computing power and storage resources. A standard solution to this problem is to offload tasks to more capable systems, such as cloud servers or data centers [6]-[8]. Offloading can improve the processing efficiency and reduce the power consumption of end devices [6]-[7], [9]. However, latency and network congestion might be induced during the offloading process [10]-[11]. In recent years, edge computing has received more and more attention due to its merits in reducing latency and bandwidth demand. Lots of effort paid to task assignment and scheduling in edge computing environments can be found in the literature [9], [12]-[13].

Many previous works resolve the scheduling problem through spectrum allocation [14]-[17]. In [14], the authors randomly assign tasks to the edge servers by a greedy method. They focus on the bandwidth demand when tasks move between edge servers and propose a solution to minimize congestion. The authors of [15] assume tasks are partitionable. Aiming at least power consumption and as many completed tasks as possible, they evaluate observed system dynamics to solve resource allocation, spectrum allocation, and power control problems. In [16], a task is partitioned into two parts. One part of them will be executed locally, and the other will be executed on the edge server. Different partition sizes and different execution durations demand different bandwidth resources. The authors schedule tasks according to tasks’ characteristics to minimize the completion time. Multi-Radio Access Technology (Multi-RAT) is employed in [17] for task partitioning. Their scheme assigns tasks to different edge servers based on the network conditions and the queue status.

In [14]-[17], only external scheduling is addressed. The cases of multicore are considered in [18]-[20] for the speed-up of the task execution. In [18], they inspect how the number of cores affects the system performance. Their scheduling allocates tasks to edge servers by weighing the availabilities of cores and tasks. The work in [19] deliberates the different power consumption in heterogeneous-core designs. The authors of [20] advocate that low delay is more demanding in remote healthy monitoring systems. They show that multicore edge servers are more suitable for real-time operation.

There are also works paying particular attention to the properties of tasks, such as workload, deadline, size, and so
forth. The context of [21] is a data center responsible for the timely delivery of data. Various tasks have different workloads and deadlines. Multiple cores operate at different clock speeds. Their scheme allocates weighted execution time to cores based on the workloads of tasks. In [22], tasks are assumed to be independent and with different priorities. With efficient management in mind, a scheme is designed to meet tasks’ QoS (Quality of Service) requirements as much as possible. A genetic algorithm is applied to multi-hop wireless networks in [23]. Data to be processed is partitioned and assigned to wireless nodes for execution. The authors define a cost function, counting for the influence of computation and communication, for improved scheduling performance. The work in [24] examines the scheduling problem in smart-grid environments. A cost function is defined to determine the target edge server for task offloading. A two-stage scheme is proposed in [25] to minimize the overall operating cost of the entire edge computing system.

Features and appeals of the approaches mentioned above are tabulated in Table I. Aiming at a comprehensive and versatile schedule scheme for edge computing, we include all three critical factors, namely partitionability, multicore, and task properties (such as deadlines or sizes), and match them with servers’ computing power, storage spaces, loadings, and networking bandwidth in this study.

### III. SYSTEM MODEL AND PROBLEM FORMULATION

#### A. SYSTEM MODEL

The context of this study is illustrated in Figure 1. It is a homogeneous network consisting of small cells, edge servers, and end devices. Edge servers are deployed in the proximity of small cells to serve end devices within a single-hop range. End devices periodically collect data and generate tasks, which will be offloaded to edge servers for execution. Each edge server has its task buffer. Results of task execution will be returned to the end device and/or passed to the cloud server for further analysis.

A scheduling scheme trying to maximize the reward of task completion needs to deliberate the varieties in the tasks’ sizes, workloads, deadline, partitionability, as well as the diversity in edge servers’ capacity, computing power, loading, etc.

#### B. PROBLEM FORMULATION

To facilitate the formulation of the problem to be addressed, a list of symbols is defined in Table II.

End devices in the environment collect data and generate tasks forming the task set, \( T \),

\[
T = \{ t_1, t_2, \ldots, t_n \}
\]

where each task \( t_i \) is denoted by a 4-tuple, as follows:

\[
t_i = (\delta_i, w_i, s_i, l_i)
\]

where \( \delta_i \) is the allowed delay; \( w_i \) is the workload; \( s_i \) is the storage requirement; and \( l_i \) is an indicator variable for task partitionability,

\[
l_i = \begin{cases} 
0, & \text{if } t_i \text{ is not partitionable} \\
1, & \text{if } t_i \text{ is partitionable} 
\end{cases}
\]

### TABLE I

**TABLE I**

| Task | Features of Related Works | Year |
|------|---------------------------|------|
|      |                           |      |
| [14] | Partitioning              | 2019 |
| [15] | Multicore Support         | 2018 |
| [16] | Task Properties           | 2018 |
| [17] |                           | 2019 |
| [18] |                           | 2019 |
| [19] |                           | 2018 |
| [20] |                           | 2018 |
| [21] |                           | 2016 |
| [22] |                           | 2017 |
| [23] |                           | 2012 |
| [24] |                           | 2019 |
| [25] |                           | 2018 |
| Ours |                           | 2020 |

### TABLE II

**TABLE II**

| Symbol | Definition |
|--------|------------|
| \( T \) | Set of all tasks |
| \( E \) | Set of all edge servers |
| \( n \) | Number of tasks |
| \( m \) | Number of edge servers |
| \( t_i \) | \( i \)-th task |
| \( \delta_i \) | Allowed delay of task \( t_i \) |
| \( w_i \) | Workload of task \( t_i \) |
| \( s_i \) | Require storage of task \( t_i \) |
| \( l_i \) | Indicator variable for task \( t_i \)’s partitionability |
| \( e_j \) | \( j \)-th edge server |
| \( S_j \) | Available storage of edge server \( e_j \) |
| \( K_j \) | Number of cores of edge server \( e_j \) |
| \( R_j \) | The processing power of edge server \( e_j \) |
| \( T_j \) | Set of tasks assigned to edge server \( e_j \) |
| \( T^* \) | Set of tasks ready to be executed |
| \( c_j \) | The reward for the in-time completion of task \( t_i \) |
| \( x_j^t \) | Indicator variable indicating if task \( t_i \) is assigned to edge server \( e_j \) |
| \( \lambda_j \) | Waiting time of task \( t_i \) on edge server \( e_j \) |
| \( \lambda^\text{exec}_j \) | The execution time of task \( t_i \) on edge server \( e_j \) |
| \( \phi_j \) | Classification of task \( t_i \) on edge server \( e_j \) |
| \( N\hat{S}_j(t_i) \) | Number of partitions of task \( t_i \) on edge server \( e_j \) |
| \( t_{f_k} \) | Starting time of execution of task \( t_i \) on edge server \( e_j \)’s \( k \)-th core |
| \( t_{f_k}^\text{th} \) | Finishing time of execution of task \( t_i \) on edge server \( e_j \)’s \( k \)-th core |
| \( t_{in_j} \) | The timing task \( t_i \) entering edge server \( e_j \) |
| \( \Phi_j \) | Set of execution costs of tasks assigned to edge server \( e_j \) |
| \( \alpha \) | The weighting factor for delay time |
| \( \beta \) | The weighting factor for execution time |
| \( \eta \) | The weighting factor for the number of cores |

A partitionable task can be divided into parts with each part assigned to a core. There are \( m \) edge servers in the environment, \( E \).
\[ E = \{ e_1, e_2, \ldots, e_m \} \]  

where each server \( e_j \) is represented by a 4-tuple, as follows:

\[ e_j = (S_j, K_j, R_j, T_j) \]  

where \( S_j \) is the storage capacity; \( K_j \) is the number of cores in \( e_j \); \( R_j \) is the computing power; and \( T_j \) is the set of tasks assigned to \( e_j \).

Assume there is a reward, \( c_i \), for the completion of task \( t_i \). The objective of our scheduling scheme is to find a subset of \( T, T^* \), to maximize the overall reward. \( T^* \) consists of \( L \) tasks which can be completed in time. Overdue tasks will be excluded. This problem can be formulated as a constrained optimization problem, as follows:

Maximize \( f(T^*) = \sum_{t_i \in T^*} c_i \)  
s.t. \[ x_j^i = \begin{cases} 
1, & \text{if } t_i \in T_j \\
0, & \text{if } t_i \notin T_j
\end{cases} \]  

(6a)

\[ \sum_{j=1}^{m} x_j^i \leq 1, \forall t_i \in T \]  

(6b)

\[ \sum_{j=1}^{n} x_j^i \cdot s_j \leq S_j, \forall e_j \in E \]  

(6c)

\[ NS_j(t_i) \leq K_j \]  

(6d)

\[ A_j(t_i) = \begin{cases} 
\text{overdue,} & \text{if } x_j^i \cdot (\lambda_{i,j}^{\text{wait}} + \lambda_{i,j}^{\text{exec}}) > \delta_i \\
\text{in time,} & \text{if } 0 < x_j^i \cdot (\lambda_{i,j}^{\text{wait}} + \lambda_{i,j}^{\text{exec}}) \leq \delta_i \\
\text{null,} & \text{if } x_j^i = 0
\end{cases} \]  

(6e)

The \( x_j^i \) in (6a) is an indicator variable. \( x_j^i \) is equal to 1 if task \( t_i \) is assigned to edge server \( e_j \), and 0 otherwise. Equation (6b) implies that a task can only be assigned to an edge server. Equation (6c) ensures that the storage capacity of an edge server cannot be exceeded. Equation (6d) specifies that a task can be partitioned into at most \( K_j \) partitions for an edge server with \( K_j \) cores.

Equation (6e) defines the condition whether task \( t_i \) can be completed in time if assigned to the edge server \( e_j \). Reward of task \( c_i \) would be assigned a value of 0 if it cannot be completed in time. \( \lambda_{i,j}^{\text{wait}} \) and \( \lambda_{i,j}^{\text{exec}} \) are the waiting time and execution time of task \( t_i \) running on an edge server \( e_j \). \( \lambda_{i,j}^{\text{wait}} \) can be evaluated according to:

\[ \lambda_{i,j}^{\text{wait}} = \min(t_{f_j}^k - t_{in_j}) \]  

(7)

where \( t_{f_j}^k \) is the commencement time of execution of task \( t_i \) on edge server \( e_j \)’s \( k \)-th core and \( t_{in_j} \) is the time task \( t_i \) entering edge server \( e_j \). \( \lambda_{i,j}^{\text{exec}} \) can be evaluated as follows:

\[ \lambda_{i,j}^{\text{exec}} = \begin{cases} 
\max(t_{f_j}^k - \min(t_{s_i}^k), & \text{if } t_i \text{ runs on multiple cores} \\
\frac{w_i}{R_j}, & \text{if } t_i \text{ run on a single core}
\end{cases} \]  

where \( t_{f_j}^k \) is the finishing time of execution of task \( t_i \) on edge server \( e_j \)’s \( k \)-th core.

Theorem 1: The scheduling problem addressed in this study, Equation (6), is an NP-complete problem.

Proof: Since (6b) denotes that a task can only be assigned to an edge server and (6d) can be set that a task can be partitioned into at most one partition for an edge server with \( K_j \) cores, we reduce the original optimization problem into a simpler one with only one constraint, as follows:

Maximize \( f(T^*) = \sum_{t_i \in T^*} c_i \)  
s.t. \[ x_j^i = \begin{cases} 
1, & \text{if } t_i \in T_j \\
0, & \text{if } t_i \notin T_j
\end{cases} \]  

(9a)

\[ \sum_{i=1}^{n} x_j^i \cdot s_j \leq S_j \]  

(9b)

This is in effect a scheduling problem considering only the capacities of edge servers if we further incorporate (6e) by setting \( c_i = 1 \) if \( x_j^i = 1 \) and \( A_j(t_i) = \text{in time} \). It can be further rewritten into the following form:

Maximize \( \sum_{i=1}^{n} c_i \cdot x_i \)  
s.t. \[ x_i \in \{0, 1\} \]  

(9a)

\[ \sum_{i=1}^{n} s_i \cdot x_i \leq S_j \]  

(9b)

The 0/1 knapsack problem is a classical NP-complete problem [26]. A typical formulation of it is given bellow:

Maximize \( \sum_{i=1}^{n} v_i \cdot p_i \)  
s.t. \[ p_i \in \{0, 1\} \]  

(10a)

\[ \sum_{i=1}^{n} w_i \cdot p_i \leq W \]  

(10b)

where \( n \) is number of items; \( v_i \) is the value of item \( i \); \( p_i \) is an indicator variable indicating if item \( i \) is packed; \( w_i \) is the size of item \( i \); and \( W \) is the capacity of the knapsack. The objective is to maximize the total value of the packing without exceeding the capacity limit.

The problem in (9) can be referred to as problem (10) following the above deductive steps. Therefore, the scheduling problem addressed in this study, Equation (6), is also an NP-complete problem.

IV. Multicore task Assignment for maximum Rewards (MAR)

To be practical, an NP-complete problem ought to be approached by heuristic methods. A two-stage scheme is developed in this section to solve the problem addressed in the last section. In the first stage, server assignment, tasks are assigned to selected edge servers, considering the servers’ capacities and accumulated workloads. In the second stage,
core assignment, tasks are assigned to chosen cores in the server. The operational flowchart of the proposed MAR scheme is given in Figure 2.

Stage 1 – Server Assignment
Algorithm 1 in Figure 3 is in charge of the offloading of tasks to edge servers. It considers edge servers’ capacities and accumulated workloads to offload tasks to adequate edge servers.

Line 1 initializes \( T_j \). Line 2 to 11 exercises the server assignment. Line 3 to 6 lists the requirements of an adequate server. Line 4 specifies that \( t_i \) and \( e_j \) are within a single-hop distance. Line 5 indicates that the inclusion of \( t_i \) does not exceed the storage capacity of \( e_j \). In addition to the conditions mentioned above, a server with the least accumulated processing workload is preferred, as shown in Line 6. If an adequate edge server, \( e_j \), can be found, \( T_j \) and \( S_j \) will be updated accordingly, as shown in Line 9 to 11. Otherwise, \( t_i \) will be removed from \( T \), as indicated in Line 7 to 8.

\[
\begin{align*}
&\text{Algorithm 1: Server Assignment} \\
&\text{Input: Set of tasks } T, \text{ set of edge servers } E \\
&\text{Output: Set of tasks in the } j\text{-th edge server } T_j, \forall j \in E \\
&1 \quad T_j = \emptyset, \forall j \in E; \\
&2 \quad \text{while } T \neq \emptyset \text{ do} \\
&3 \quad \text{find } e_j \in E \text{ such that} \\
&4 \quad \quad 1) e_j \text{ is a single hop away from } t_i; \\
&5 \quad \quad 2) s_i + \sum_{g=1}^{n} x^g \cdot s_g \leq S_j; \\
&6 \quad \quad 3) (\sum_{g=1}^{n} x^g \cdot \frac{w_g}{r_j}) \text{ has the minimum value}; \\
&7 \quad \text{if no suitable } e_j \text{ can be found then} \\
&8 \quad T \leftarrow T \setminus \{t_i\}; \\
&9 \quad \text{else} \\
&10 \quad T_j \leftarrow T_j \cup \{t_i\}, T \leftarrow T \setminus \{t_i\}; \\
&11 \quad \text{Update } S_j; \\
&12 \quad \text{return } T_j;
\end{align*}
\]

FIGURE 3. Algorithm 1 - server assignment.

\[ \text{Lemma 1: Assume there are } n \text{ tasks in } T \text{ and } m \text{ edge servers in } E. \text{ Algorithm 1 is of time complexity } O(n \cdot m). \]

\[ \text{Proof: The initialization process in Line 1 is of time complexity } O(m). \text{ Line 2 to 11 iterate for each of } n \text{ tasks.} \]

Within the loop, Line 3 to 6 iterate for each of \( m \) edge servers. Thus, the time complexity of Algorithm 1 is of \( O(m) + O(n \cdot m) = O(n \cdot m) \).

Stage 2 – Core Assignment
The core assignment is illustrated in Figure 4. The edge server evaluates the execution cost of tasks assigned to it with Function 1. Execution cost decides the order of execution. The lower the execution cost is, the higher the execution priority is. A task might be divided into parts depending on its partitionability. The proposed scheme assigns a standalone task to a core or partitions it into subtasks to multiple cores, one subtask for each core, minimizing the overall execution time. The ultimate goal is to accomplish as many tasks as possible to receive a maximal total reward.
**Output:** Set of execution costs in the $j$-th edge server $Φ_j$

1. $Φ_j = \emptyset$;
2. **for** $t_i \in T_j$ **do**
3.  **if** $I_l = 1$ **then**
4.    $ϕ_l = \left(\alpha \cdot δ_1 + \beta \cdot \frac{w_i}{R_j}, (K_j \cdot η)\right) \cdot \frac{1}{ε_i}$;
5.  **else**
6.    $ϕ_l = \left(\alpha \cdot δ_1 + \beta \cdot \frac{w_i}{R_j}, (K_j \cdot η)\right) \cdot \frac{1}{ε_i}$;
7.  $Φ_j \leftarrow Φ_j \cup \{ϕ_l\}$
8.  **return** $Φ_j$;

**FIGURE 5.** Function for the evaluation of execution cost.

Due to the variety in tasks’ workloads and deadlines, a dedicated scheme is required to conduct the core assignment once tasks had been assigned to a particular edge server. To decide the order of task execution, our scheme calculates the execution cost of individual tasks. Lower cost implies a higher priority of execution. Also, the associated reward can be easier to be obtained.

$Φ_j = \{ϕ_1, \cdots, ϕ_{l_j}\}$ is the set of execution cost, $ϕ_l$, of tasks assigned to edge server $e_j$, where $ϕ_{l_j}$ is the number of tasks assigned to $e_j$. The evaluation of $ϕ_l$ depends on its partitionability, as follows:

$$ϕ_l = \begin{cases} 
\left(\alpha \cdot δ_1 + \beta \cdot \frac{w_i}{R_j}, (K_j \cdot η)\right) \cdot \frac{1}{ε_i}, & \text{if } I_l = 1 \\
\left(\alpha \cdot δ_1 + \beta \cdot \frac{w_i}{R_j}, (K_j \cdot η)\right) \cdot \frac{1}{ε_i}, & \text{if } I_l = 0
\end{cases}$$  \hspace{1cm} (11)

Equation (11) takes into account tasks’ allowed delays, workloads, rewards, as well as servers’ processing power, number of cores. $α$ is a weighting factor for the allowed delay; $β$ is a weighting factor for the execution time; $η$ is a weighting factor for the number of cores; $K_j$ is the number of cores of edge server $e_j$. An important implication of (11) is that a partitionable task is assigned with a higher cost since it can be partitioned and assigned to multiple cores for execution. The methodology is to endow non-partitionable tasks with higher priorities of execution to increase their chance of completion to earn higher total rewards. The cost of execution can be evaluated with Function 1 given in Figure 5. The time complexity of Function 1 is of $O(L_j) = O(n)$.

After execution costs of all tasks are evaluated, Algorithm 2 – core assignment, is called for core dispatch. Tasks in the queue are sorted according to their execution costs in ascending order and processed one by one. A task will be dropped if it is not possible to be completed by its deadline.

**Algorithm 2: Core Assignment**

**Input:** Set of tasks in $j$-th edge server $T_j$, set of edge servers $E$

**Output:** The execution schedule

1. **for** $e_j \in E$ **do**
2.  $Φ_j = \text{Execution\_Cost}(T_j)$;
3.  **while** $T_j \neq \emptyset$ **do**
4.    $i = \text{arg min}_{\substack{t_j \in Φ_j}} (ϕ_i)$;
5.    $ϕ_{\text{wait}} = \min(t_{\text{ts},i}) - t_{\text{in},i}$;
6.    $\text{exec\_bound} = δ_i - ϕ_{\text{wait}}$;
7.    $ϕ_{\text{exec}} = \frac{w_i}{R_j}$;
8.    **if** $ϕ_{\text{exec}} \leq \text{exec\_bound}$ **then**
9.      Task $t_i$ is scheduled to a core with the lightest loading to compute;
10.  $A_j(t_i) = \text{in\_time}$;
11.  **else**
12.    $ϕ_{\text{exec}} > \text{exec\_bound}$
13.    $ϕ_{\text{exec}} = \max(t_{\text{ts},i}) - \min(t_{\text{ts},i})$;
14.    **if** $ϕ_{\text{exec}} < \text{exec\_bound}$ **then**
15.      Task $t_i$ is scheduled to multiple cores to compute;
16.    $A_j(t_i) = \text{in\_time}$;
17.  **else**
18.    $A_j(t_i) = \text{overdue}$;
19.  **end**
20.  $T_j \leftarrow T_j \setminus \{t_i\}$;
21.  **Update** $S_j$;

**FIGURE 6.** Algorithm 2 – core assignment.

**Lemma 2:** Assume there are $L_j$ tasks in $T_j$, $n$ tasks in $T$, and $m$ edge servers in $E$. Algorithm 2 is of time complexity $O(m \cdot n^2)$.

**Proof:** Line 1 to 22 iterate for each of the $m$ edge servers. Line 2 is of time complexity $O(L_j)$, as explained previously. Line 3 to 20 iterate for each of the $L_j$ tasks. The time complexity of Line 4 is $O(L_j)$. Thus, Algorithm 2 is of time complexity $O \left(m \cdot \left(L_j + L_j^2\right)\right) = O(m \cdot L_j^2) = O(m \cdot n^2)$. ■

**V. SIMULATIONS AND ANALYSIS**
A series of simulations were conducted using MATLAB to verify the feasibility and effectiveness of the proposed MAR scheme. MATLAB object classes for end devices and edge servers were created to simulate the functionalities and behaviors of their counterparts in the real world. The simulations platform is a Windows 10 PC with an i5 CPU, 16 GB DDR RAM, and 500GB SSD. For each experiment, we report the average results of 30 runs.

The completion ratio of tasks, which denotes the number of tasks completed in time divided by the total number of tasks, is a performance indicator of primary concern. A high completion ratio also implies a high satisfaction rate.

| Parameter                  | Value                             |
|----------------------------|-----------------------------------|
| Network size               | 1550×1550 m²                      |
| Number of end devices      | [500, 3000]                       |
| Number of small cells      | 105                               |
| Radius of small cells      | 100 m                             |
| Inter-site distance        | 150 m                             |
| Computation workload of each task | 270000 – 330000 clock cycles [14] |
| Storage of each task       | 2880 – 3520 bits [14]             |
| Processing power of each edge server’s core | 50 MCPS [14] |
| Number of cores in each edge server | 4                                 |
| α                          | 0.2                               |
| β                          | 0.8                               |

A. SIMULATION SETTINGS

The parameter settings for the simulations are given in TABLE III. In the simulated environment [27], there are 105 small cells residing in a 1550 m × 1550 m region. The radius of a small cell is 100 m. Small cells are 150 m apart from each other. An edge server with four cores is deployed at each small cell. It is assumed that every end-device is single-hop with its neighbor edge server, and edge servers can communicate with each other via small cell. The computing power of each core is 50 MCPS (Million Cycles Per Second). There are 500 to 3,000 end devices in the environment. Tasks generated by end devices have their workload from 270,000 to 330,000 clock cycles and storage requirements between 2,880 and 3,520 bits. We also consider the distribution of end devices. There are two scenarios in our simulations, as shown in Figure 7 and Figure 8. Figure 7 is a typical scenario for uniform distribution. For non-uniform distribution, there are ten hotspots out of 105 small cells, as shown in Figure 8. 60% of the end devices reside within the coverage of hotspots. In other words, each hotspot cell covers 6% of the end devices.

B. COMPARATIVE STUDY AND ANALYSIS

To figure out the relative merits and limitations, we have the following schemes included in our comparative study:

• First-Come-First-Serve (FCFS): Tasks are executed in the order they arrive at the edge servers. A core will be occupied till the completion of a task.

• Shortest Task First (STF): Scheduling is done according to tasks’ workloads. Tasks with smaller workloads are preferred. The task with the smallest workload will be scheduled first. The dispatch of core begins with the least loaded core.

• Delay Priority Scheduling (DPS): The scheduling is done based on allowed delays of tasks. A task with a smaller allowed delay is given a higher priority in execution. The task with the smallest allowed delay will be scheduled foremost.
• Green Greedy Algorithm (GGA) [24]: A cost function considering tasks’ workload, edge servers’ computing capability, and power consumption is defined. The total workloads of servers are used to decide the target server in task offloading.

• Proposed: The MAR scheme proposed in this article.

Here we assume that the reward for each task is the same. Then, the maximal overall reward is equivalent to the maximal number of completed tasks. Hence, the task complete ratio, the percentage of completed tasks, is our primary concern. We shall examine how the complete ratio is affected by different system settings, such as the number of end devices, the distribution of end devices, the setting of $\eta$, etc. We use the task generation rate to indicate the percentage of end devices that generate tasks in an epoch.

Figure 9 shows how the completion ratio changes as the number of end devices vary in Scenario 1. When the system is lightly loaded, all schemes perform well. The proposed scheme prevails as the system load gets heavier. We have a performance ranking list as follows: The Proposed MAR scheme, DPS, GGA, STF, and finally FCFS. We can see a similar pattern in subsequent simulations. GGA, STF, and FCFS do not consider the allowed delays. Tasks can become overdue and be dropped. GGA defines a cost function to decide offloading server. Therefore, it has better performance than STF and FCFS. There are fewer dropped tasks with DPS. However, DPS considers only allowed delay, so its performance is still inferior to that of the proposed MAR scheme.

![Figure 9. Completion ratio vs. the number of end devices (Scenario 1, Generation rate: 100%).](image)

We turn our attention to the effect of $\eta$, the weighting factor for the number of cores. The completion ratio of all tasks and partitionable (independent) tasks are given in Figure 11. The proposed scheme is designed to postpone the execution of partitionable tasks since a partitionable task can be divided and assigned to multiple cores for a shorter execution time. According to the design, a larger $\eta$ will favor non-partitionable tasks and a smaller $\eta$ will favor partitionable tasks and. The improved overall completion ratio with a greater $\eta$ value is at the cost of a lower completion ratio for partitionable tasks.

![Figure 11. Completion ratios (Scenario 2, Generation rate: 80%, # of end devices: 1,500).](image)

We now have a closer look into the system behavior within hotspots. As shown in Figure 12, the performance pattern resembles that in Figure 10, whereas with a shift down along the y-axis and steeper declination. It implies that a large number of end devices in a hotspot increases the amount of dropped tasks.
Figure 13 illustrates how system performance is affected by the distribution of end devices. When there are 20% of end devices located within hotspots, all schemes have a completion ratio above 85%. Note that as the percentage increases to 80%, the system performance drops rapidly. It is because a large number of end devices residing within hotspots would saturate the system capacity. Anyhow, the proposed MAR scheme can still maintain a completion ratio as high as 85%.

The effect of the different number of cores is given in Figure 14. For the same number of end devices, a more-core setting leads to a higher completion ratio. Because partitionable tasks can be allocated to multiple cores for shortening the execution time. The chance of overdue is also decreased accordingly. As the number of end devices increases, the performance difference among the different number of cores becomes increasingly significant. For 3,000 end devices, a six-core setting can maintain a completion ratio of 90%, while the completion ratio of a dual-core setting drops to 30%.

VI. CONCLUSIONS

With the rise of the Internet of Things, the amount of collected data grows explosively. Due to the insufficient computing power of the end devices, tasks should be offloaded to edge servers for the sake of timely response and balanced loading. However, due to the uneven distribution of devices in the real environment, hot spots may occur. There could be too many devices under some servers, which may cause some tasks to expire due to failing to be scheduled properly.

To solve this problem, this paper proposes a task scheduling mechanism. First, we weigh each edge computing server’s storage space and workload to ascertain an adequate target server for the task offloading. Next, the internal task scheduling, the core assignment, is executed. We distinguish the priority of different tasks by calculating the execution cost of each task. The execution cost factors the allowed delay, the workload, the number of processor cores, and processing power. The priority of non-partitionable tasks will be slightly higher than partitionable tasks. Partitionable tasks can reduce the chance of overdue by partitioning them into sub-tasks and mapping them to multiple cores.

In our simulations, we observe the task completion rate under different conditions. Simulation results corroborate that the proposed scheme outperforms FCFS, STF, DPS, and GGA in maintaining a higher task completion ratio under different situations and conditions.

REFERENCES

[1] Z. Zhang and S. Li, “A survey of computational offloading in mobile cloud computing,” 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, pp. 81-82, 2016.
[2] S. Naveen and M. R. Koune, “Key technologies and challenges in IoT edge computing,” 2019 3rd International Conference on IoT in social, mobile, analytics and cloud, pp. 61-65, 2019.
[3] Y. Lee, W. Kim, K. Moon, and K. Lim, “A mobile edge computing device to support data collecting and processing from IoT,” 2019 International Conference on Electronics, Information, and Communication, pp. 1-3, 2019.
[4] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, Mobile Edge Computing - A Key Technology towards 5G, ETSI, vol. 11, 2015.

[5] N. Abbas, Y. Zhang, A. Taherkordi and T. Skeie, “Mobile edge computing: A survey,” IEEE Internet of Things Journal, vol. 5, no. 1, pp. 450-465, 2018.

[6] K. Kumar, J. Liu, Y. Lu, and B. K. Bhargava, “A survey of computation offloading for mobile systems,” Mobile Networks and Applications, vol. 18, no. 1, pp. 129-140, 2013.

[7] M. R. Rahimi, J. Ren, C. H. Liu, A. V. Vasilakos, and N. Venkatasubramanian, “Mobile cloud computing: A survey state of art and future directions,” Mobile Networks and Applications, vol. 19, no. 2, pp. 133-143, 2014.

[8] K. Kumar and Y. Lu, “Cloud computing for mobile users: Can offloading computation save energy?” IEEE Computer, vol. 43, no. 4, pp. 51-56, 2010.

[9] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, Z. Chen, P. S. Pillai, and M. Satyanarayanan, “Quantifying the impact of edge computing on mobile applications,” ACM SIGOPS Asia-Pacific Workshop on Systems, vol. 5, pp. 1-8, 2016.

[10] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for VM-based cloudlets in mobile computing,” IEEE Pervasive Computing, vol. 8, no. 4, pp. 14-23, 2009.

[11] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, “Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system,” IEEE Transactions on Computers, vol. 65, no. 12, pp. 3702-3712, 2016.

[12] X. Chen, L. Jiao, W. Li, and X. Fu, “Efficient multi-user computation offloading for mobile-edge cloud computing,” IEEE/ACM Transactions on Networking, vol. 24, no. 5, pp. 2795-2808, 2016.

[13] Y. Mao, J. Zhang, and K. B. Letaief, “Dynamic computation offloading for mobile-edge computing with energy harvesting devices,” IEEE Journal on Selected Areas in Communications, vol. 34, no. 12, pp. 3590-3605, 2016.

[14] Y. Sahni, J. Cao, and L. Yang “Data-aware task allocation for achieving low latency in collaborative edge computing,” IEEE Internet of Things Journal, vol. 6, no. 2, pp. 3512-3524, 2019.

[15] F. Wei, S. Chen, and W. Zou, “A greedy algorithm for task offloading in mobile edge computing system,” China Communications, vol. 15, no. 11, pp. 149-157, 2018.

[16] S. Wang, C. Pan, and C. Yin, “Joint heterogeneous tasks offloading and resource allocation in mobile edge computing systems,” 2018 10th International Conference on Wireless Communications and Signal Processing, Hangzhou, pp. 1-6, 2018.

[17] Z. Jing, Q. Yang, M. Qin, and K. S. Kwak, “Long term max-min fairness guarantee mechanism: Adaptive task splitting and resource allocation in MEC-enabled networks,” 2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications, pp. 1-6, 2019.

[18] A. Gamatje, G. Devic, G. Sassatelli, S. Bernabovi, P. Naudin, and M. Chapman, “Towards energy-efficient heterogeneous multicore architectures for edge computing,” IEEE Access, vol. 7, pp. 49474-49491, 2019.

[19] S. Yoo, “Leveraging multicore for mobile edge computing,” 2018 International Conference on Information Networking, pp. 869-874, 2018.

[20] M. A. Disi, H. Djeloutou, A. Amira, and F. Bensaali, “The accuracy and efficacy of real-time compressed ECG signal reconstruction on a heterogeneous multicore edge-device,” 2018 21st Euromicro Conference on Digital System Design, pp. 458-463, 2018.

[21] M. Zhang, S. Wang, G. Yuan, Y. Li, and Z. Qian, “Energy-efficient real-time task allocation in a data center,” IEEE International Conference on Internet of Things and IEEE Green Computing and Communications and IEEE Cyber, Physical and Social Computing and IEEE Smart Data, pp. 680-687, 2016.

[22] Y. Song, S. S. Yau, R. Yu, X. Zhang, and G. Xue, “An approach to QoS-based task distribution in edge computing networks for IoT applications,” 2017 IEEE International Conference on Edge Computing, pp. 32-39, 2017.

[23] Y. Jin, J. Jin, A. Gluhak, K. Moessner, and M. Palaniswami, “An intelligent task allocation scheme for multi-hop wireless networks,” IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 3, pp. 444-451, 2012.

[24] J. Yao, Z. Li, Y. Li, J. Bai, J. Wang, and P. Lin, “Cost-efficient tasks scheduling for smart grid communication network with edge computing system,” 2019 15th International Wireless Communications & Mobile Computing Conference, pp. 272-277, 2019.

[25] Y. Zhang, X. Chen, Y. Chen, Z. Li, and J. Huang, “Cost efficient scheduling for delay-sensitive tasks in edge computing system,” 2018 IEEE International Conference on Services Computing, pp. 73-80, 2018.

[26] S. Martello and P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, 1990.

[27] Y. Lin, “Multi-layer load balancing with D2D relay assist in dense small cell networks,” Master Thesis, Department of Computer Science and Engineering National Sun Yat-sen University, 2019.