Chen et al. Journal of Cloud Computing (2023) 12:107
https://doi.org/10.1186/s13677-023-00477-9

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

An intelligent approach of task offloading for dependent services in Mobile Edge Computing

Jie Chen[1], Yajing Leng[1] and Jiwei Huang[1*]

Abstract

With the growing popularity of Internet of Things (IoT), Mobile Edge Computing (MEC) has emerged for reducing the heavy workload at the multi-cloud core network by deploying computing and storage resources at the edge of network close to users. In IoT, services are data-intensive and event-driven, resulting in extensive dependencies among services. Traditional task offloading schemes face significant challenges in the IoT scenario with service dependencies. To this end, this paper proposes an intelligent approach for minimizing latency and energy consumption which jointly considers the task scheduling and resource allocation for dependent IoT services in MEC. Specifically, we establish the system model, communication model as well as computing model for performance evaluation by fully considering the dependent relationships among services, and an optimization problem is proposed for minimizing the delay and energy consumption simultaneously. Then, we design a layered scheme to deal with the service dependencies, and present detailed algorithms to intelligently obtain optimal task scheduling and resource allocation policies. Finally, simulation experiments are carried out to validate the effectiveness of the proposed scheme.

Keywords Mobile Edge Computing, Internet of Things, Offloading decision, Resource allocation, Delay, Energy consumption

Introduction

With the rapid development of smart mobile devices and Internet of Things (IoT), various IoT services show explosive growth [1]. IoT services enrich people's lives, but they also put forward higher requirements for hardware resources of IoT devices, such as computing resources, storage resources and battery life [2]. With the dramatic increase in the computational complexity of services in IoT, only relying on the limited computing capacity of IoT devices often can not guarantee the timely execution of various tasks [3]. Given the architecture of IoT devices and the trend of battery development, these problems will also be difficult to solve in the future [4]. Mobile Cloud Computing (MCC) is considered an effective solution to the above problems. In MCC, migrating the data processing and storage of IoT services to the multi-cloud for computing provides users with powerful data computing and storage capabilities. In addition, it reduces energy consumption of IoT devices and prolongs battery life. However, the explosive growth of IoT devices and data transfers poses enormous challenges to MCC [5]. Excessive network load makes it impractical for IoT devices to transfer all the massive data generated to multi-cloud for centralized processing. In addition, some new IoT services requiring extremely low latency are emerging in large numbers. Offloading all these IoT services over the core network to a remote multi-cloud can result in high latency [6]. Therefore, MCC is not suitable for IoT services with very low latency requirements.

*Correspondence:
Jiwei Huang
huangjw@cup.edu.cn
[1] Beijing Key Laboratory of Petroleum Data Mining, China University of Petroleum, Beijing, China

Mobile Edge Computing (MEC) is a new computing mode, which focuses on providing users with certain Internet Technology (IT) and cloud computing capabilities at the edge of the mobile network [7]. In 2014, the European Telecommunications Standards Institute (ETSI) proposed the concept of Mobile Edge Computing [8]. MEC provides users with powerful computing, storage, network and communication resources near the edge of their mobile network (such as base stations, wireless access points, etc.), which reduces the latency of tasks and improves the Quality of user Experience (QoE) [9]. In addition, it can alleviate users' concerns about privacy leaks and ensure data security by eliminating the need to transfer data to multi-cloud for processing [10].

Computational Offloading is one of the key technologies of MEC, which mainly includes the following two aspects [11]:

(1) Offloading Decision: Deciding whether the task is handled on the local device or on an MEC server;
(2) Resource Allocation: Allocating the resources needed to process tasks, including computing resources, communication resources, etc.

However, due to the heterogeneity of IoT devices and MEC servers and the diversity of IoT services, it is difficult to obtain a common computational offloading strategy [12, 13]. Therefore, it is necessary to design an appropriate computational offloading strategy based on different MEC environments, IoT services, and optimization objectives.

Modern IoT services often consist of multiple tasks with dependencies [14]. Dependency means that there is a priority constraint between tasks, such as a task cannot be started until all tasks with a higher priority than it have been processed. In MEC, when tasks with dependencies are processed on IoT devices and MEC servers separately or on different MEC servers, data transmission across devices will usually occur. Complex dependencies and commucication between IoT services makes it more difficult to realize optimal offloading decisions and resource allocation schemes [15, 16]. So, in the existing studies, few gave consideration to both offloading decisions and resource allocation to optimize task latency and energy consumption simultaneously. This paper considers task offloading and resource allocation of dependent IoT services in user-oriented MEC scenarios to optimize latency and energy consumption. Different from the previous works, our contributions are as follows.

(1) To solve the problem of task dependency, this paper presents a layered algorithm based on topological sorting, which layers tasks so that there is no dependency between tasks in the same layer after layering, and then gets the optimal offloading decision and resource allocation of all tasks in the layer in turn.
(2) To decide whether a task in a layer is offloaded or not, we use the weighted sum of latency and energy consumption to define the cost of local and edge calculations for tasks. To determine the optimal offloading decision and resource allocation scheme, we calculate and compare the minimum cost of a task computed locally and offloaded to MEC server. Simulation experiments were conducted to verify the effectiveness of the algorithm in optimizing latency and energy consumption.

The rest of this paper is organized as follows. In the next section, we discuss the releated works. Then, we present the system model and problem formulation. Based on the system model, the following section gives a layered computational offloading algorithm for the minimum overhead. Next, we conduct experiments to verify the effectiveness of the proposed algorithm. Finally, we conclude the paper in the last section.

Related work

In the computation offloading of MEC, most of the existing researches focus on reducing task latency or energy consumption by studying reasonable offloading decision.

By offloading computing tasks from IoT devices to MEC servers with rich computing resources, latency of tasks can be significantly reduced [17]. Real time applications such as AR, VR and the Internet of Vehicles that are sensitive to time delay require ultra-low time delay to provide continuous services. Therefore, there is a lot of research in MEC that focuses on reducing task delay through task offloading. The literature [18] studied an MEC system that allows computing tasks to be executed in parallel on IoT devices and MEC servers with the goal of reducing the latency of computing tasks. An efficient one-dimensional search algorithm is proposed. Although this scheme has significant effect in reducing delay, there are still some defects. For example, this scheme is not applicable to dependent tasks, and does not consider the signaling cost of terminal receiving feedback from MEC server. The literature [19] studied the problem of task offloading in 5G ultra dense networks and establishes a problem to minimize the total delay of all tasks under the constraint of residual power of IoT devices, which is a mixed integer nonlinear programming problem. On this basis, the author proposes an effective computational offloading scheme, which can reduce task delay by 20% compared with random offloading and uniform offloading schemes. However, the final offloading strategy of this scheme depends too much on the given initial task offloading strategy.

Because rapid energy consumption poses a major obstacle in the contemporary network [19], there are also a lot of researches aimed at reducing energy consumption in MEC. Due to the limited battery life of IoT devices, most relevant researches focus on reducing the energy consumption of IoT devices. In order to reduce the total energy consumption of the MEC system, the literature [20] considers joint optimisation of offloading decisions and wireless resource allocation. The joint optimisation problem is difficult to solve due to its non-convexity and NP-hard property. To reduce the solution complexity, the original problem is transformed into a two-layer optimisation problem. Specifically, the optimal transmission power and subcarrier allocation can be obtained by the Lagrange multiplier method for a given initial task offloading strategy. And on the basis of obtaining the optimal transmission power and subcarrier allocation, the optimal offloading strategy is solved by using the Hungarian algorithm.

In MEC, higher transmission rate requires higher power at the transmitter and receiver, which will reduce task delay, but also lead to more energy consumption, and vice versa [21, 22]. Therefore, it is also an important research direction of computational offloading to comprehensively consider the latency and energy consumption to improve the Quality of Service (QoS) and user experience of MEC system. The literature [23] investigates task offloading and resource allocation in a multi-user MEC system using time division multiple access as the uplink transmission mechanism, which shares a single MEC server. The optimal resource allocation problem is programmed as a convex optimization problem that minimizes the overhead (weighted sum of delay and energy consumption) by considering two cases of limited and unlimited MEC server resources, and the optimal offloading is obtained by solving the problem.

Modern IoT services usually consist of multiple dependent tasks [24]. The literature [25] investigates the offloading of sequentially dependent tasks and concurrent tasks. For sequential dependent tasks, the authors clarify that successive offloading of tasks is required to reduce the overall latency. A violence-based search approach is then used to find the starting and ending tasks that need to be offloaded. For concurrent tasks, the task dependencies are degraded to a tree, and then clusters of tasks are offloaded to minimize latency based on the idea of load balancing. For more general task dependencies (where there are both sequential and concurrent tasks), concurrent tasks are first aggregated into virtual tasks and then the offloading decision for the task is found by using the method of offloading sequentially dependent tasks. For virtual tasks that are decided to be processed on the IoT device, they are then offloaded by using the offloading method for concurrent tasks. Finally, simulation results show that the method is twice as fast as the baseline method and achieves 85% of the performance of the optimal solution.

System model and problem formulation

Task dependency model

As shown in Fig. 1, suppose that an IoT device has a service that needs to be processed. Service generated by the IoT device consists of $|V|$ dependent tasks. Each task can be processed on the IoT device or offloaded to an MEC server through the wireless network. Directed Acyclic Graph (DAG) $G = (V, E)$ is used to model the service dependency [26]. Wherein, node $v_i \in V$ represents the *ith* task generated by the IoT device, while the edge $< v_i, v_j > \in E$ represents the dependency between tasks (task v_j can only be started after task v_i has completed processing and v_j has received the output of task v_i), where v_i is called the

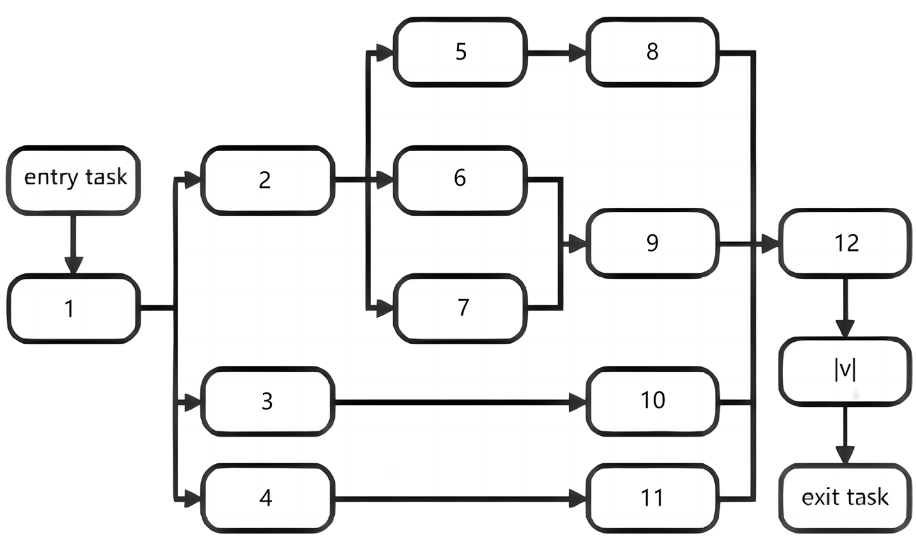

Fig. 1 An example of service dependency

precursor task of v_j, and v_j is called the successor task of v_i. For a given DAG, a task without any precursor is called an entry task, and a task without any successor is called an exit task. For the *ith* task generated by the IoT device, a quaternion in the form of $v_i = (C_i, D_i, I_i, O_i)$ is used for modeling, where C_i represents the calculation amount of v_i, that is, the number of CPU cycles required to process v_i, I_i represents the amount of input data that v_i receives from all its precursor tasks, O_i represents the amount of output data of v_i after processing, and D_i represents the amount of data of v_i. To ensure the first task is executed from the IoT device and the final processing results can be returned to the IoT device, a virtual entry task v_{entry} and a virtual exit task v_{exit} are added to the DAG [27]. Among them, $C_{entry} = C_{exit} = 0$, $D_{entry} = D_{exit} = 0$, $I_{entry} = 0$, $O_{entry} = I_1$, $I_{exit} = O_{|V|}$, $O_{exit} = O_{|V|}$, and v_{entry} and v_{exit} can only be processed on the local IoT device and cannot be offloaded to an MEC server. For multi entry tasks or multi exit tasks, adding a virtual entry task and an exit task can also simplify the DAG into single entry task and single exit task for easy solution. Therefore, the total number of tasks n is:

$$n = |V| + 2 \tag{1}$$

Communication model

When a task is offloaded to an MEC server or two dependent tasks are processed on different devices respectively (For example, task v_i is processed on the IoT device and its successor task is processed on an MEC server), data transmission between the IoT device and an MEC server through the wireless network is involved, and an appropriate communication model needs to be established to analyze the communication overhead.

(1) Offload task v_i to an MEC server When the task v_i is offloaded to an MEC server, the uplink data transmission rate r_i^o is:

$$r_i^o = Blog_2(1 + \frac{p_i^o h_i}{\sigma_i^2}) \tag{2}$$

Wherein, B represents the channel bandwidth, h_i represents the channel gain between the base station and the IoT device, σ_i^2 represents the noise power, p_i^o represents the transmission power that v_i is offloaded from the IoT device, $p^{min} \le p_i^o \le p^{max}$, p^{min} and p^{max} are respectively the minimum and maximum transmission power of the IoT device. Correspondingly, $r^{min} \le r_i^o \le r^{max}$, where, $r^{min} = Blog_2(1 + \frac{p^{min} h_i}{\sigma_i^2})$, $r^{max} = Blog_2(1 + \frac{p^{max} h_i}{\sigma_i^2})$. The offloading delay required for offloading task v_i to an MEC server is:

$$t_i^o = \frac{D_i}{r_i^o} \tag{3}$$

Wherein, $\frac{D_i}{r^{max}} \le t_i^o \le \frac{D_i}{r^{min}}$. The energy consumption for transmisson of the IoT device required for offloading task v_i to an MEC server is:

$$e_i^o = p_i^o t_i^o = \frac{D_i \sigma_i^2}{r_i^o h_i}(2^{\frac{r_i^o}{B}} - 1) \tag{4}$$

(2) Upload the output of the task v_i to an MEC server When a task v_i is processed on the IoT device and a successor task of the task v_i needs to be offloaded to an MEC server, the IoT device needs to transmit the output of v_i to an MEC server through the wireless network. The uplink data transmission rate r_i^u of uploading the output of task v_i is:

$$r_i^u = Blog_2(1 + \frac{p_i^u h_i}{\sigma_i^2}) \tag{5}$$

Wherein, $p^{min} \le p_i^u \le p^{max}$, correspondingly, $r^{min} \le r_i^u \le r^{max}$. The transmission delay required for uploading the output of task v_i to an MEC server is:

$$t_i^u = \frac{O_i}{r_i^u} \tag{6}$$

Wherein, $\frac{O_i}{r^{max}} \le t_i^u \le \frac{O_i}{r^{min}}$. The transmission energy consumption of the IoT device required for uploading the output of the task v_i to an MEC server is:

$$e_i^u = p_i^u t_i^u = \frac{D_i \sigma_i^2}{r_i^u h_i}(2^{\frac{r_i^u}{B}} - 1) \tag{7}$$

(3) Download the output of the task v_i to the IoT device When a task v_i is processed on an MEC server, and a successor task of v_i needs to be processed on the IoT device, the MEC server needs to send the output of v_i back to the IoT device through the wireless network. Assuming that the download rate is constant at r^d, the transmission delay of downloading the output result of the task v_i is:

$$t_i^d = \frac{O_i}{r^d} \tag{8}$$

Computational model

Because tasks can be processed on the IoT device or offloaded to MEC servers, two different models need to be considered.

(1) Local computing model The IoT device uses Dynamic Frequency Scaling (DFS) technology to reduce energy consumption [28]. Therefore, the energy consumption of the IoT device during local computing can be reduced by adjusting the CPU frequency. Therefore, the computing delay when task v_i is processed on the IoT device can be expressed as follows:

$$t_i^l = \frac{C_i}{f_i^l} \tag{9}$$

Wherein, f_i^l represents the calculation frequency of the IoT device when processing task v_i and f^{max} represents the maximum frequency of the IoT device processor. Because $f_i^l \leq f^{max}$, so $t_i^l \geq t_i^{min}$, wherein $t_i^{min} = \frac{C_i}{f^{max}}$. The computing energy consumption when task v_i is processed on the IoT device can be expressed as [29]:

$$e_i^l = \kappa f_i^3 t_i^l = \kappa \frac{C_i^3}{(t_i^l)^2} \tag{10}$$

Wherein, $\kappa > 0$ refers to the energy efficiency parameter.

(2) Edge computing model The calculation delay when task v_i is processed on an MEC server can be expressed as:

$$t_i^e = \frac{C_i}{f^e} \tag{11}$$

Wherein, f^e is the processor frequency of an MEC server.

Problem formulation

Binary variable a_i is used to indicate whether the task v_i is offloaded ($a_i = 1$ indicates that the task v_i is offloaded to an MEC server, $a_i = 0$ indicates that the task v_i is processed on the IoT device). During the whole offloading process, the following restrictions need to be met:

(1) For $\forall < v_i, v_j > \in E$, the moment t_j^s when the successor task v_j starts processing cannot be earlier than the moment when the precusor task v_i completes processing and the output of the task v_i is received by v_j;
(2) Each task is either offloaded to an MEC server or processed on the local device;
(3) Tasks on the same device must be processed serially. For example, if task v_i and task v_j are offloaded to an MEC server for processing, one of them must wait for the other to finish processing before it can be processed.

In order to optimize task delay and energy consumption of the IoT device at the same time, referring to [20], the optimization goal is considered as the weighted sum of all task delay and local device energy consumption. In order to facilitate the solution, the startup time of the virtual entry task v_{entry} is set as 0, and the delay of all tasks can be expressed as:

$$T = \sum_{i=1}^{n} t_i^f \tag{12}$$

Wherein, t_i^f is the completion time of the task v_i.

The energy consumption of IoT devices includes three parts: Firstly, locally computed energy consumption; Secondly, the energy consumption of offloading tasks which need to be processed on MEC servers; Thirdly, the energy consumption for uploading output of tasks which are processed on the local device and whose successor tasks are offloaded to MEC servers. Therefore, the energy consumption of the local device can be expressed as:

$$E = \sum_{i=1}^{n} [(1 - a_i)e_i^l + a_i e_i^o] + \sum_{<v_i, v_j> \in E} (1 - a_i) a_j e_i^u \tag{13}$$

Therefore, the problem can be planned as follows:

$$\begin{aligned} & min \omega_1 T + \omega_2 E \\ & s.t. \begin{cases} C1 : t_i^s + (1 - a_i)t_i^l + a_i t_i^e + a_i(1 - a_j)t_i^d + (1 - a_i)a_j t_i^u \leq t_j^s, \forall < v_i, v_j > \in E \\ C2 : t_i^s - t_j^s + \chi(3 - (1 - a_i) - (1 - a_j) - x_{i,j}) \geq t_j^l, \forall v_i, v_j \in V \\ C3 : t_i^s - t_j^s + \chi(3 - a_i - a_j - x_{i,j}) \geq t_j^e, \forall v_i, v_j \in V \\ C4 : x_{i,j} + x_{j,i} = 1, \forall v_i, v_j \in V \\ C5 : p^{min} \leq p_i^o, p_i^u \leq p^{max}, \forall v_i \in V \\ C6 : 0 < f_i^l \leq f^{max}, \forall v_i \in V \\ C7 : a_i \in \{0, 1\}, \forall i = 1, 2, \cdots, n \\ C8 : a_0 = a_n = 0 \\ C9 : x_{i,j} \in \{0, 1\}, \forall v_i, v_j \in V \\ C10 : t_i^s \geq 0, \forall v_i \in V \end{cases} \end{aligned} \tag{14}$$

Wherein, ω_1 and ω_2 are weighting factors, and $\omega_1 + \omega_2 = 1$, χ is a large number. The constraint C1 guarantees that for $\forall < v_i, v_j > \in E$, the time t_j^s when the task v_j starts possessing is not earlier than the time when the task v_i finishes processing and the output of the task v_i is received by v_j. Constraints C2 and C4 ensure that tasks processed on the IoT device must be processed serially. Constraints C3 and C4 ensure that tasks processed on an MEC server must be processed serially. The constraint C5 ensures that the transmission power of the IoT device must be between its maximum power and minimum power. The constraint C6 ensures that the calculation frequency of the IoT device cannot exceed its maximum calculation frequency. Constrain C7 ensures that tasks can only be offloaded to MEC servers or processed on the local device. Constraint C8 ensures that the first task and the last task must be processed on the local device. Constraint C9 guarantees that $x_{i,j}$ can only take 0 or 1. C10 ensures that the start time of task processing cannot be negative. Obviously, this is a mixed integer nonlinear programming problem and an NP-hard problem [27].

Layered computational offloading algorithm

In order to solve the above problems, this section proposes a layered offloading algorithm to minimize the overhead. Firstly, the tasks of the IoT device are layered according to the DAG of dependent services by using the layered algorithm based on topological sorting. After the layering, there is no dependency between the tasks in the same layer. Then the task of each layer is offloaded in order and the optimal resource allocation scheme is obtained.

Concept definition

To facilitate the description and analysis, the following concepts are defined first.

Concept about delay

(1) Actual computing delay T_i^{exec} of the task v_i The actual processing delay of the task v_i is defined as the actual computing delay of the task v_i on the device (the local device or an MEC server) after the offloading decision. When the task v_i is processed on the local device ($a_i = 0$), its actual computing delay can be expressed as:

$$T_i^{exec} = t_i^l \tag{15}$$

When the task is processed on an MEC server ($a_i = 1$), the actual computing delay can be expressed as:

$$T_i^{exec} = t_i^e \tag{16}$$

(2) The time RT_i when the task v_i can be started Referring to [20], the concept of the startable time RT_i of the task v_i on the device (the local device or an MEC server) is introduced. RT_i is defined as the earliest time when the task v_i receives all precursor task outputs and the device is idle at the same time. The time RT_i^l of the task v_i on the local device can be expressed as follows :

$$RT_i^l = max\{T_l^{idle}, max_{j\in pred(i)}\{AFT_j + a_j t_j^d\}\} \tag{17}$$

Wherein, T_l^{idle} represents the idle time of the IoT device, $pred(i)$ represents the collection of all the precusor tasks of the task v_i, and AFT_j represents the time when the task v_j is actually completed. The startable time RT_i^e of the task v_i on the MEC server can be expressed as:

$$RT_i^e = max\{T_e^{idle}, max_{j\in pred(i)}\{AFT_j + (1-a_j)t_j^u + t_i^o\}\} \tag{18}$$

Wherein, T_e^{idle} represents the idle time of the MEC server.

(3) The time AST_i when the task v_i is actually started The time AST_i is defined as the time when the task v_i is actually started to be processed during the processing of all tasks. Because the actual starting time of the task v_i is not earlier than the startable time, so there must be $AST_i \geq RT_i$.

(4) The time AFT_i when the task v_i is actually completed The actual completion time of a task v_i is equal to the sum of its actual start time and its actual computing delay. Therefore, the actual completion time AFT_i of the task v_i can be expressed as follows:

$$AFT_i = AST_i + T_i^{exec} \tag{19}$$

Concept about energy consumption

(1) Energy consumption E_i^l of the task v_i processed on the IoT device When the task v_i is processed on the IoT device, the energy consumption of the local device is only the computing energy consumption for processing the task v_i. So the energy consumption E_i^l can be expressed as follows:

$$E_i^l = e_i^l \tag{20}$$

(2) Energy consumption E_i^e of the task v_i processed on an MEC server When the task v_i is processed on an MEC server, the energy consumption of the local device is the energy consumption of offloading the task v_i and the energy consumption of transmitting the output of all its precursor tasks processed on the local IoT device to MEC servers. Therefore, the energy consumption E_i^e can be expressed as follows:

$$E_i^e = e_i^o + \sum_{j\in pred(i)} (1-a_j)e_j^u \tag{21}$$

Layered algorithm

Topological sorting is a standard algorithm for solving the linear sorting of DAG vertices [30]. In the vertex sequence generated by topological sorting, for $\forall <i,j> \in E$, task v_i is before task v_j. Topological sorting is used to sort tasks to ensure that all the precursor tasks of the task v_j are ahead of them. However, this method can only produce one task sequence. Therefore, referring to the idea of topological sorting, an algorithm is designed to layer dependent tasks, so that there is no dependency between tasks in the same layer after layering.

In this paper, a layering algorithm is proposed based on topological sequences. The procedure of the algorithm is shown in Algorithm 1. The adjacency table (adjList) is used to store the DAG nodes and their dependencies. Line 2 of the algorithm initializes a queue for subsequent operations. Lines 3-4 of the algorithm put the entry task v_{entry} into the queue. Line 8 indicates that as long as the queue is not empty, the current length of the queue (*len*) is obtained, which is the number of tasks in the next layer. In line 10-20, *len* tasks leave the queue in order and are put into the set (*curLayer*) that stores tasks of the current layer. When each task leaves the queue, the indegree of all its successor tasks will be reduced by 1. If the indegree of a successor task becomes 0, the successor task will be put into the queue. After processing all the tasks of the current layer, put the set (*curLayer*) into a list storing layers (*layerList*), and then continue to process the tasks of the next layer. After all tasks are processed, return layering results (*layerList*).

```
Input: adjList, entryTask
Output: layerList
 1: funciton DividingLayer(adjList)
 2: Create a new queue: Q
 3: if entryTask!=NULL then
 4:     entryTask enters Q
 5: end if
 6: Create a new list: layerList
 7: while Q is not empty do
 8:     Get the length of Q: len
 9:     create a new set: curLayer
10:     while len > 0 do
11:         a task (curTask) leave the queue
12:         for every successor task (sucTask) of curTask do
13:             indegree of sucTask minus 1
14:             if indegree of sucTask is 0 then
15:                 sucTask enters Q
16:             end if
17:         end for
18:         put curTask into curLayer
19:         len=len-1
20:     end while
21:     put curLayer into layerList
22: end while
23: return layerlist
24: end function
```

Algorithm 1 Task layering algorithm

Cost analysis of processing tasks

The cost $Cost_i$ of processing task v_i is divided into two parts: one is the delay required for processing tasks (computational latency and transmission latency); The other is the energy consumption of the IoT device (processing energy consumption or transmission energy consumption) when processing tasks. Therefore, $Cost_i$ can be expressed as:

$$Cost_i = \omega_1 AFT_i + \omega_2 E_i \tag{22}$$

Next, we will analyze the cost of the task processed on the local IoT device and offloaded to an MEC server separately according to whether the task is offloaded.

(1) The cost of task v_i when processed locally When the task v_i is calculated on the local IoT device, according to the previous analysis, $Cost_i$ can be expressed as a function of one variable, with its variable being local computational latency t_i^l, so $Cost_i$ can be expressed as follows:

$$Cost_i(t_i^l) = Cost_i^l(t_i^l) = \omega_1(RT_i^l + t_i^l) + \omega_2\kappa\frac{C_i^3}{(t_i^l)^2} = \omega_1 RT_i^l + g_i(t_i^l) \tag{23}$$

Wherein, $g_i(x) = \omega_1 x + \frac{\omega_2\kappa C_i^3}{x^2}$. Obviously, $g_i(x)$ decreases monotonically in the range of $0 < x \leq \sqrt[3]{\frac{2\omega_2\kappa C_i^3}{\omega_1}}$ and increases monotonically in the range of $x \geq \sqrt[3]{\frac{2\omega_2\kappa C_i^3}{\omega_1}}$. Use t_i^{opt} to represent the corresponding t_i^l when the local computing overhead $Cost_i^l(t_i^l)$ is optimal. If $\sqrt[3]{\frac{2\omega_2\kappa C_i^3}{\omega_1}} \leq t_i^{min}$, then $t_i^{opt} = t_i^{min}$. Otherwise, $t_i^{opt} = \sqrt[3]{\frac{2\omega_2\kappa C_i^3}{\omega_1}}$. Therefore, $t_i^{opt} = max(\sqrt[3]{\frac{2\omega_2\kappa C_i^3}{\omega_1}}, t_i^{min})$. Therefore, Algorithm 2 is designed to obtain the minimum cost of task v_i processed on the local IoT device and the corresponding optimal latency of local computing. After obtaining the optimal local latency of the task v_i, the optimal computing resource allocated by the local IoT device can be obtained from the equation (9).

Input: a task v_i
Output: the minimun overhead $cost_i^{opt}$, optimal latency t_i^{opt}
function LocalComputingCost(v_i)
$t_i^{min} = \frac{C_i}{f^{max}}$
$T = \sqrt[3]{\frac{2\omega_2\kappa C_i^3}{\omega_1}}$
$t_i^{opt} = max(T, t_i^{min})$
$cost_i^{opt} = Cost_i^l(t_i^{opt})$
return t_i^{opt},$cost_i^{opt}$
end function

Algorithm 2 Solving for the minimum cost of local computation and its corresponding optimal latency algorithm

(2) The cost of task v_i when offloaded According to the previous analysis, when the task v_i is offloaded to an MEC server for calculation, $Cost_i$ can be expressed as a binary function, with its variables being t_i^o and t_j^u, and its specific form can be expressed as follows:

$$Cost_i(t_i^o, t_j^u) = \omega_1(RT_i^e + t_i^e) + \omega_2[e_i^o + \sum_{j \in pred(i)} (1-a_j)e_j^u] \tag{24}$$

Because the binary function increases the difficulty of solving, offloading task v_i and transmitting the output of its precursor tasks processed on the local IoT device are considered to use the same transmission rate (i.e. $r_j^u = r_i^o = r_i$). At this time, $Cost_i$ is converted into a unary function only related to the transmission rate r_i :

$$\begin{aligned} Cost_i(r_i) =& \omega_1(RT_i^e + t_i^e) + \omega_2[\frac{D_i\sigma^2}{r_ih_i}(2^{\frac{r_i}{B}} - 1) + \sum_{j \in pred(i)} (1-a_j)\frac{O_j\sigma^2}{r_ih_j}(2^{\frac{r_i}{B}} - 1)] \\ =& \omega_1(max\{T_e^{idle}, max_{j \in pred(i)}\{AFT_j + \frac{(1-a_j)O_j + D_i}{r_i}\}\}) + c_i + b_if(\frac{r_i}{B}) \end{aligned} \tag{25}$$

Wherein, $f(x) = \frac{2^x - 1}{x}$, $c_i = \omega_1 t_i^e$, $b_i = \frac{\omega_2\sigma_i^2}{B}(\frac{D_i}{h_i} + \sum_{j \in pred(i)} \frac{(1-a_j)Oj}{h_j})$. Obviously, when $x \geq 0, f(x)$ is monotonically increasing.

If $r_i \geq R_i^1$, then:

$$Cost_i(r_i) = Cost_i^{e1}(r_i) = \omega_1 T_e^{idle} + c_i + b_if(\frac{r_i}{B}) \tag{26}$$

Wherein, $R_i^1 = max_{j \in pred(i)}(\frac{D_i + (1-a_j)O_j}{T_e^idle - AFT_j})$.At this time, $Cost_i(r_i)$ increases monotonically.

If $r_i < R_i^1$, then:

$$\begin{aligned} Cost_i(r_i) =& \omega_1 max_{j \in pred(i)}\{AFT_j + \frac{(1-a_j)O_j + D_i}{r_i}\} + c_i + b_if_i(\frac{r_i}{B}) \\ =& \omega_1 max\{MAFT_i^e, max_{j \in pred(i) \wedge a_j = 0}\{AFT_j + \frac{O_j}{r_i}\}\} + h_i(\frac{r_i}{B}) \end{aligned} \tag{27}$$

Wherein, $MAFT_i^e = max_{j \in pred(i) \wedge a_j = 1}(AFT_j)$, $h_i(x) = \frac{a_i}{x} + b_if_i(x) + c_i$, $a_i = \frac{\omega_1 D_i}{B}$, $h_i^{'}(x) = \frac{b_i(ln2 \cdot x \cdot 2^x - 2^x + 1) - a_i}{x^2}$. Obviously, when $x \geq 0$, $ln2 \cdot x \cdot 2^x - 2^x + 1$ is monotonically increasing, so, there are only three possible values of $h_i(x)$ in the range of $[x_1, x_2]$:

(1) If $h_i^{'}(x)$ is always less than or equal to 0, $h_i(x)$ decreases monotonically, and the optimal value of $h_i(x)$ at this time is obtained at $x = x_2$;
(2) If $h_i^{'}(x)$ is always greater than or equal to 0, $h_i(x)$ increases monotonically, and the optimal value of $h_i(x)$ at this time is obtained at $x = x_1$;
(3) When $h_i^{'}(x)$ is less than or equal to 0 at first and then greater than or equal to 0, $h_i(x)$ is a single-peaked function, and 0.618 method can be used to search for the optimal value.

So, the above analysis of solving for the minimum value of h(x) and its corresponding x can be summarized as the function $OPTIMAL(h(x), x_1, x_2)$. Wherein, input of OPTIMAL consists of a monotone or single-peaked function $h(x)$, lower bound x_1 and upper bound x_2. Output of OPTIMAL are the minimum value h^{opt} of $h(x)$ and its corresponding optimal value x^{opt} of x. Specific steps of function OPTIMAL are as follows. Firstly, Solve for the derivative function $h'(x)$ of $h(x)$. Secondly, three cases are considered. If $h'(x_2) \leq 0$, then $x^{opt} = x_2$ and $h^{opt} = h(x_2)$. If $h'(x_1) \geq 0$, then $x^{opt} = x_1$ and $h^{opt} = h(x_1)$. If $h'(x_1) < 0$ and $h'(x_2) > 0$, then h^{opt} and x^{opt} can be obtained by 0.618 method.

If $R_i^2 \leq r_i \leq R_i^1$, then:

$$Cost_i(r_i) = Cost_i^{e2}(r_i) = \omega_1 MAFT_i^e + h_i(\frac{r_i}{B}) \tag{28}$$

Wherein, $R_i^2 = max_{j \in pred(i) \wedge a_j = 0}(\frac{O_j}{MAFT_i^e - AFT_j})$. The minimum cost is obtained by function OPTIMAL.

If $r_i \leq min(R_i^1, R_i^2)$, then:

$$\begin{aligned} Cost_i(r_i) =& \omega_1 max_{j \in pred(i) \wedge a_j = 0}(AFT_j + \frac{O_j}{r_i}) + \omega_1 t_i^e + \omega_1 \frac{D_i}{r_i} \\ &+ \frac{\omega_2\sigma^2}{B}(\frac{D_i}{h_i} + \sum_{j \in pred(i)} (1-a_j)\frac{O_j}{h_j})f(\frac{r_i}{B}) \end{aligned} \tag{29}$$

Obviously, for AFT_j, $\frac{O_j}{r_i}$ is negligible. Therefore, $Cost_i(r_i)$ can be expressed as:

$$Cost_i(r_i) = Cost_i^{e3}(r_i) = \omega_1 MAFT_i^l + h_i(\frac{r_i}{B}) \tag{30}$$

Wherein, $MAFT_i^l = max_{j \in pred(i) \wedge a_j = 0 (AFT_j)}$. Obtain the minimum cost by function OPTIMAL.

Through the above analysis, the minimum cost of task v_i processed on an MEC server and the corresponding optimal transmission rate can be obtained. After obtaining the optimal transmission rate of the IoT device, the optimal transmission power allocated by the local device can be obtained by equation (2) or (5). The procedure of the algorithm is shown in Algorithm 3. Line 2 calculates R_i^1 and R_i^2 . Lines 3-4 show that when $R_i^1 < r^{min}$, there must be $R_i^1 < r_i$, so $Cost_i(r_i) = Cost_i^{e1}(r_i)$, monotonically increasing. Lines 8-10 show that when $R_i^2 < r^{min} \leq R_i^1 < r^{max}$, if $R_i^1 \leq r_i$, then $Cost_i(r_i) = Cost_i^{e1}(r_i)$, monotonically increasing, so $r_i^{opt,1} = R_i^1$; If $r_i < R_i^1$, then $Cost_i(r_i) = Cost_i^{e2}(r_i)$, the minimum cost $Cost_i^{e2}(r_i^{opt,2})$ and its corresponding optimal transmission rate $r_i^{opt,2}$ can be obtained by function OPTIMAL. Combining these two cases, the minimum cost and its corresponding optimal transmission rate can be obtained when $R_i^2 < r^{min} \leq R_i^1 < r^{max}$, that is $r_i^{opt} = argmin(Cost_i^{e1}(r_i^{opt,1}), Cost_i^{e2}(r_i^{opt,2}))$. Similarly, lines 13-16 consider the minimum cost when $r^{min} \leq R_i^2 \leq R_i^1 < r^{max}$. Lines 18-20 consider the minimum cost when $r^{min} \leq R_i^1 < min(R_i^2, r^{max})$. Line 25 considers the minimum cost when $R_i^2 \leq r^{min} \leq r^{max} \leq R_i^1$. Lines 28-30 consider the minimum cost when $r^{min} \leq R_i^2 \leq r^{max} \leq R_i^1$. Line 32 considers the minimum cost when $r^{max} \leq min(R_i^2, R_i^1)$.

Input: a task v_i
Output: the minimun cost $cost_i^{opt}$, optimal transmisson rate r_i^{opt}
funciton EdgeComputingCost(v_i)
$R^1 = max_{j \in pred(i)}(\frac{D_i+(1-a_j)O_j}{T_e^{idle}-AFT_j})$, $R^2 = max_{j \in pred(i) \wedge a_j=0}(\frac{O_j}{MAFT_i^e-AFT_j})$
if $R^1 < r^{min}$ **then**
    $r_i^{opt} = r^{min}$, $cost_i^{opt} = Cost_i^{e1}(r_i^{opt})$
else
    if $R^1 < r^{max}$ **then**
        if $R^2 < r^{min}$ **then**
            $r_1 = R^1$, $cost_1 = Cost_i^{e1}(r_1)$
            $r_2, cost_2 = OPTIMAL(Cost_i^{e2}, r^{min}, R^1)$
            $r_i^{opt} = argmin(cost_1, cost_2), cost_i^{opt} = min(cost_1, cost_2)$
        else
            if $R^2 < R^1$ **then**
                $r_1 = R^1, cost_1 = Cost_i^{e1}(r_1)$
                $r_2, cost_2 = OPTIMAL(Cost_i^{e2}, R^2, R^1)$
                $r_3, cost_3 = OPTIMAL(Cost_i^{e3}, r^{min}, R^2)$
                $r_i^{opt} = argmin(cost_1, cost_2, cost_3), cost_i^{opt} = min(cost_1, cost_2, cost_3)$
            else
                $r_1 = R^1$, $cost_1 = Cost_i^{e1}(r_1)$
                $r_2, cost_2 = OPTIMAL(Cost_i^{e3}, r^{min}, R^1)$
                $r_i^{opt} = argmin(cost_1, cost_2), cost_i^{opt} = min(cost_1, cost_2)$
            end if
        end if
    else
        if $R^2 \le r^{min}$ **then**
            $r_i^{opt}, cost_i^{opt} = OPTIMAL(Cost_i^{e2}, r^{min}, r^{max})$
        else
            if $R^2 \le r^{max}$ **then**
                $r_1, cost_1 = OPTIMAL(Cost_i^{e2}, R^2, r^{max})$
                $r_2, cost_2 = OPTIMAL(Cost_i^{e3}, r^{min}, R^2)$
                $r_i^{opt} = argmin(cost_1, cost_2), cost_i^{opt} = min(cost_1, cost_2)$
            else
                $r_i^{opt}, cost_i^{opt} = OPTIMAL(Cost_i^{e3}, r^{min}, r^{max})$
            end if
        end if
    end if
end if
return $r_i^{opt}, cost_i^{opt}$
end function

Algorithm 3 Solving for the minimum cost of edge computation and its corresponding optimal transmission rate algorithm

Layered computational offloading algorithm

From the above analysis, the costs of the task computed on the local IoT device or offloaded to an MEC server are obtained separately, which is related to resource allocation scheme. Here we propose an algorithm based on comparing the minimum cost of the task v_i processed on the IoT device and on an MEC server separately to determine the offloading decision and resource allocation of the task v_i. The main steps of the algorithm are as follows:

(1) Call Algorithm 1 to layer tasks;
(2) For all tasks in the first layer, Algorithm 4 is called to determine whether each task in the same layer is offloaded or not in turn;
(3) Use the same method in step 2 for subsequent layers until all layers are processed.

The procedure of the intra-layer offloading algorithm is shown in Algorithm 4. For all tasks in a given layer, line 2 of Algorithm 4 indicates that the tasks are arranged in ascending order according to the completion time of each task's latest precursor task. Lines 3-13 indicate that the minimum cost of the task processed on the local device and processed on an MEC server are calculated separately in order . If the local computing overhead is less than the offloading overhead, the task is processed locally and the optimal computing frequency of the IoT device is obtained. Then, the next idle time of the local IoT device is updated. Otherwise, the task will be offloaded, and the optimal transmission power of the IoT device will be obtained, and then the next idle time of the MEC server will be updated.

Input: a task list (layerList) of one layer
funciton OffloadingInLayer(layerList)
Arrange the tasks within the layer in ascending order by the value $max_{j \in pred(i)}(AFT_j)$
for every task v_i in layerList **do**
  $t^{local}, cost^{local} = LocalComputingCost(v_i)$
  $r^{edge}, cost^{edge} = EdgecomputingCost(v_i)$
  if $cost^{local} < cost^{edge}$ **then**
    $a_i = 0, AFT_i = T_i^{idle} + t^{local}$
    $e_i = \kappa \frac{C_i^3}{(t^{local})^2}$
    $T_l^{idle} = AFT_i$
  else
    $a_i = 1, AFT_i = T_e^{idle} + \frac{\sum_{j \in pred(i) \wedge a_j = 0} O_j + D_i}{r^{edge}} + t_i^e$
    $e_i = e_i^o + \sum_{j \in pred(i) \wedge a_j = 0} e_j^u$
    $T_e^{idle} = AFT_i$
  end if
end for
end function

Algorithm 4 Intra-layer offloading decision and resource allocation algorithm

Performance evaluation

Setup

The two types of task dependency models shown in Fig. 2 are used for simulation experiments. In Fig. 2(a), tasks can only be executed sequentially, and there is only one task in each layer. In Fig. 2(b), each layer has five tasks, and these five tasks are the precursor tasks of all tasks in the next layer. In addition, the number of tasks in these two types of task dependency models will be set to 10, 20, 30, 40 and 50, respectively, to evaluate the performance of the proposed algorithm under different task numbers and task dependencies.

The bandwidth of the wireless channel is set to 5 MHz. Consider that the white noise power and channel gain are the same when all tasks are offloaded, wherein, $\sigma_1^2 = \cdots = \sigma_{|\nu|}^2 = \sigma^2 = 10^{-10} W$, $h_1 = \cdots = h_{|\nu|} = h$. For h, the path fading model is used for modeling [28], and the specific form is as follows:

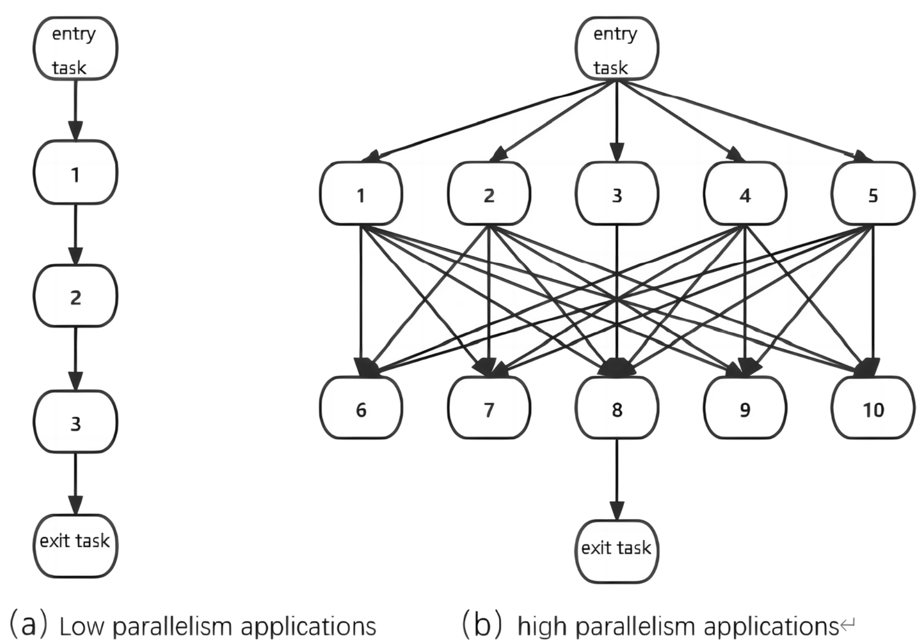

Fig. 2 Low and high parallelism task dependency model

$$h = A_d(\frac{3 \times 10^8}{4\pi f_c d_M})^{d_e} \quad (31)$$

Wherein, A_d represents antenna gain, f_c represents carrier frequency, d_M represents the distance between IoT device and MEC server, and d_e represents path fading factor. For IoT devices, the maximum calculation frequency f^{max} is set to 0.5 GHz, and the minimum transmission power p^{min} and maximum transmission power p^{max} are set to 0.1 W and 0.5 W respectively. For computing tasks, The required CPU cycle follows the uniform distribution of $[100, 200] \times 10^6$ Cycles, the data size follows the uniform distribution of [2, 5] MB, and the computing output follows the uniform distribution of [2, 5] KB. For the MEC server, its calculation frequency f^e is set to 5 GHz. The energy efficiency parameter κ is set to 10^{-27}. Other simulation parameters are shown in Table 1.

Comparison experiments

With reference to [31, 32], we compare the proposed algorithm with three other offloading baseline strategies and the earliest completion time offloading strategy proposed in [33]:

(1) Local computing strategy: The local computing strategy does not involve task offloading. All computing tasks are processed on the local IoT device, and the computing frequency of the IoT device is randomly determined from 0GHz to 0.5GHz.
(2) Edge computing strategy: In the edge computing strategy, all IoT devices offload their computing tasks randomly to a nearby MEC server for processing, and randomly determine the transmission power of IoT devices from 0.1W to 0.5W.
(3) Random offloading strategy: In the random offloading strategy, the decision whether task is offloaded or not and which edge server to offload is determined randomly. The calculation frequency is randomly determined from 0GHz to 0.5GHz and transmission power of the IoT device is randomly determined from 0.1W to 0.5W.
(4) The earliest completion time offloading strategy: Calculate the average calculation and communication cost of each task and determine the processing order of the task, assign the task to the processor with the minimum completion time in turn according to the processing order. The calculation frequency is randomly determined from 0GHz to 0.5GHz and transmission power of the IoT device is randomly determined from 0.1W to 0.5W.

Table 1 some other parameters

parameter	value
Number of tasks $\|V\|$	{10,20,30,40,50}
Bandwidth B	5MHz
White noise power σ^2	$10^{-10}W$
Antenna gain A^d	4.11
carrier frequency f_c	915MHz
Path fading factor d_e	2.6
Distance between mobile device and MEC server d_M	30

The task latency and energy consumption of IoT devices are used as performance indicators to evaluate the offloading performance of five computational offloading strategies for low parallelism and high parallelism dependent IoT services.

Figure 3 compares the performance of five strategies when $\omega_1 = 0.9$ and $\omega_2 = 0.1$. At this time, more attention is paid to the task latency rather than the energy consumption of IoT devices. It can be seen that, for both low-parallelism dependent services and high-parallelism dependent services, the computational offloading strategy proposed in this paper always obtains lower latency than other strategies. For low-parallelism dependent services, the performance of random offloading strategy is between local computing strategy and edge computing strategy, while for high-parallelism dependent services, the performance of random offloading strategy is better than local computing strategy and edge computing strategy. This is because, for low-parallelism dependent services, tasks can only be executed sequentially, that is, the next task can only be started after the precursor task has been processed. Therefore, the performance of the random offloading strategy must be somewhere between the two. For high-parallelism dependent services, when random offloading strategy is adopted, tasks can be processed in parallel to a certain extent. For local computing strategy (or edge computing strategy), the task can only be processed after the IoT device (or MEC server) finishes processing the precursor task. That is, due to the limitations of the processor, the task can only be serial. Therefore, random offloading strategy is superior to local computing strategy and edge computing strategy.

Figure 4 compares the performance of the five strategies when $\omega_1 = 0.1$ and $\omega_2 = 0.9$. At this time, more attention is paid to the energy consumption of IoT devices rather than the task latency. It can be seen that, for both low-parallelism dependent services and high-parallelism dependent services, the computational offloading strategy proposed in this paper always obtains lower energy consumption than other strategies. It can be seen that the energy consumption of each computational offloading strategy for low parallelism and high parallelism dependent services is roughly the same.

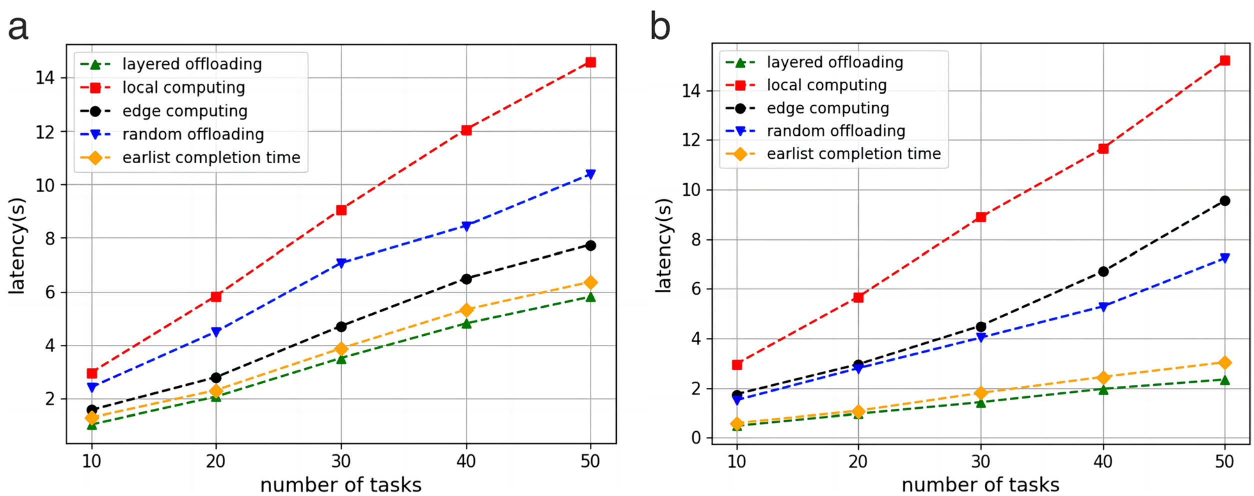

Fig. 3 The performance of five strategies when $\omega_1 = 0.9$ and $\omega_2 = 0.1$

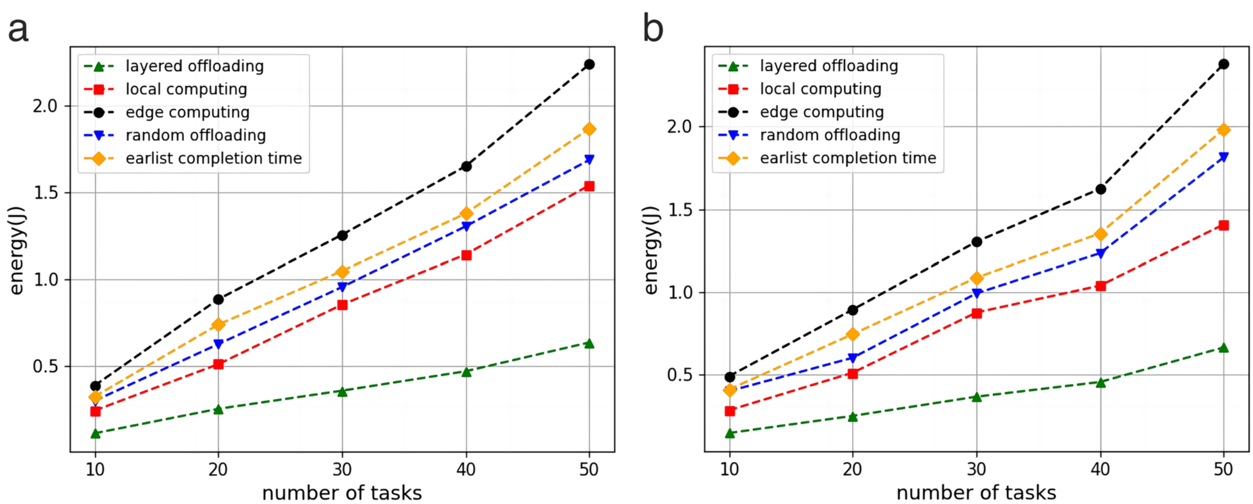

Fig. 4 The performance of five policies when $\omega_1 = 0.1$ and $\omega_2 = 0.9$

This is because, at this time, more attention is paid to energy consumption, and the dependency between IoT services has less impact on the energy consumption than on the task latency of IoT devices.

Conclusion

In this paper, task scheduling and resource allocation are comprehensively considered to optimize the latency and energy consumption for dependent IoT services in MEC. We design a computational offloading algorithm based on layering tasks by dependencies, to get the optimal task offloading scheduling and resource allocation scheme. Simulation results show that the proposed algorithm is significantly better than other comparison algorithms in reducing latency and energy consumption.

For our future work, we will consider further improvements in future research:

(1) In the system model, this paper assumes a constant download rate to facilitate the analysis and optimization of the task latency. However, task results are typically transmitted over wireless networks, which have a time-varying transmission rate. We will further consider the varying download rate and design its corresponding optimization scheme in our future work.

(2) In the optimization goal, this paper uses the weighted sum of latency and energy consumption as the optimization goal to comprehensively optimize the latency and energy consumption. However, how to determine the weight value is a difficult problem to solve. In the next step, we can consider using multi-objective optimization methods, such as Pareto, multi-objective particle swarm optimization, etc.

Acknowledgements
The authors would like to thank the anonymous reviewers for their insightful comments and suggestions on improving this paper.

Authors' contributions
Jie Chen designed the modeling approach and the algorithm, and wrote the paper. Yajing Leng designed system model and carried out the experiments. Jiwei Huang conceived the initial idea proofread the manuscript. The authors read and approved the final manuscript.

Funding
This work is supported by Beijing Nova Program (No. Z201100006820082), National Natural Science Foundation of China(No. 61972414).

Availability of data and materials
The datasets used during the current study are available from the corresponding author on reasonable request.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 30 December 2022 Accepted: 25 June 2023
Published online: 18 July 2023

References

1. Chen Y, Gu W, Xu J, et al (2022) Dynamic task offloading for digital twin-empowered mobile edge computing via deep reinforcement learning. China Commun
2. Chen Y, Zhao J, Wu Y et al (2022) Qoe-aware decentralized task offloading and resource allocation for end-edge-cloud systems: A game-theoretical approach. IEEE Trans Mob Comput. https://doi.org/10.1109/TMC.2022.3223119
3. Satyanarayanan M (1996) Fundamental challenges in mobile computing. In: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing. pp 1–7
4. Satyanarayanan M (1993) Mobile computing. Computer 26(9):81–82
5. Huang J, Wan J, Lv B, Ye Q et al (2023) Joint computation offloading and resource allocation for edge-cloud collaboration in internet of vehicles via deep reinforcement learning. IEEE Syst J. https://doi.org/10.1109/JSYST.2023.3249217
6. Chen H, Qin W, Wang L (2022) Task partitioning and offloading in iot cloud-edge collaborative computing framework: a survey. J Cloud Comput 11(1):1–19
7. Chen Y, Hu J, Zhao J, Min G (2023) Qos-aware computation offloading in leo satellite edge computing for iot: A game-theoretical approach. Chin J Electron
8. Chen Y, Xing H, Ma Z, et al (2022) Cost-efficient edge caching for noma-enabled iot services. China Commun
9. Huang J, Lv B, Wu Y et al (2022) Dynamic admission control and resource allocation for mobile edge computing enabled small cell network. IEEE Trans Veh Technol 71(2):1964–1973. https://doi.org/10.1109/TVT.2021.3133696
10. Tran-Dang H, Kim DS (2021) Frato: fog resource based adaptive task offloading for delay-minimizing iot service provisioning. IEEE Trans Parallel Distrib Syst 32(10):2491–2508
11. Hai LA, Sz B, Zc A, Hl C, Lw D (2020) A survey on computation offloading modeling for edge computing - sciencedirect. J Netw Comput Appl 169:102781
12. Huang J, Gao H, Wan S et al (2023) Aoi-aware energy control and computation offloading for industrial iot. Future Generation Comput Syst 139:29–37
13. Chen Y, Zhao J, Zhou X, et al (2023) A distributed game theoretical approach for credibility-guaranteed multimedia data offloading in mec. Inf Sci
14. Liao Y, Shou L, Yu Q, Ai Q, Liu Q (2020) An intelligent computation demand response framework for iiot-mec interactive networks. IEEE Netw Lett 2(3):154–158
15. Chen J, Chen P, Niu X, Wu Z, Xiong L, Shi C (2022) Task offloading in hybrid-decision-based multi-cloud computing network: a cooperative multi-agent deep reinforcement learning. J Cloud Comput 11(1):1–17
16. Chen Y, Zhao J, Hu J, et al (2023) Distributed task offloading and resource purchasing in noma-enabled mobile edge computing: Hierarchical game theoretical approaches. ACM Trans Embed Comput Syst
17. Almutairi J, Aldossary M (2021) A novel approach for iot tasks offloading in edge-cloud environments. J Cloud Comput 10(1):1–19
18. Liu J, Mao Y, Zhang J, Letaief KB (2016) Delay-optimal computation task scheduling for mobile-edge computing systems. In: 2016 IEEE international symposium on information theory (ISIT). IEEE, pp 1451–1455
19. Chen M, Hao Y (2018) Task offloading for mobile edge computing in software defined ultra-dense network. IEEE J Sel Areas Commun 36(3):587–597
20. Cheng K, Teng Y, Sun W, Liu A, Wang X (2018) Energy-efficient joint offloading and wireless resource allocation strategy in multi-mec server systems. In: 2018 IEEE international conference on communications (ICC). IEEE, pp 1–6
21. Muñoz O, Pascual-Iserte A, Vidal J (2013) Joint allocation of radio and computational resources in wireless application offloading. In: 2013 Future Network & Mobile Summit. IEEE, pp 1–10
22. Li K, Zhao J, Hu J et al (2022) Dynamic energy efficient task offloading and resource allocation for noma-enabled iot in smart buildings and environment. Build Environ. https://doi.org/10.1016/j.buildenv.2022.109513
23. You C, Huang K (2016) Multiuser resource allocation for mobile-edge computation offloading. In: 2016 IEEE Global Communications Conference (GLOBECOM). IEEE, pp 1–6
24. Zhao W, Chellappa R, Phillips PJ, Rosenfeld A (2003) Face recognition: A literature survey. ACM Comput Surv (CSUR) 35(4):399–458
25. Jia M, Cao J, Yang L (2014) Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing. pp 352–357
26. Lin X, Wang Y, Xie Q, Pedram M (2014) Energy and performance-aware task scheduling in a mobile cloud computing environment. In: 2014 IEEE 7th international conference on cloud computing. IEEE, pp 192–199
27. Vu TT, Van Huynh N, Hoang DT, Nguyen DN, Dutkiewicz E (2018) Offloading energy efficiency with delay constraint for cooperative mobile edge computing networks. In: 2018 IEEE Global Communications Conference (GLOBECOM). IEEE, pp 1–6
28. Ji J, Zhu K, Yi C, Wang R, Niyato D (2020) Joint resource allocation and trajectory design for uav-assisted mobile edge computing systems. In: GLOBECOM 2020-2020 IEEE Global Communications Conference. IEEE, pp 1–6
29. Wang Y, Min S, Wang X, et al (2016) Mobile-edge computing: partial computation offloading using dynamic voltage scaling[J]. IEEE Trans Commun 64(10):4268–4282
30. Prabhumoye S, Salakhutdinov R, Black AW (2020) Topological sort for sentence ordering. arXiv preprint arXiv:2005.00432
31. Li J, Gao H, Lv T, Lu Y (2018) Deep reinforcement learning based computation offloading and resource allocation for mec. In: 2018 IEEE Wireless communications and networking conference (WCNC). IEEE, pp 1–6

32. Zhang Y, Niyato D, Wang P (2015) Offloading in mobile cloudlet systems with intermittent connectivity. IEEE Trans Mob Comput 14(12):2516–2529
33. Huang Q, Ang P, Knowles P, Nykiel T, Tverdokhlib I, Yajurvedi A, Dapolito IV P, Yan X, Bykov M, Liang C, et al (2017) Sve: Distributed video processing at facebook scale. In: Proceedings of the 26th Symposium on Operating Systems Principles. pp 87–103

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.