2025 International Conference on Intelligent Computing
July 26-29, Ningbo, China
https://www.ic-icc.cn/2025/index.php

Research on Edge-Device Collaborative Task Offloading for Dependent Tasks

Bo Peng, Xinxin Chen, Qiang Li(✉), Li Yan and Xinyu Zhang

School of Information and Control Engineering, Southwest University of Science and Technology. 17, 69121 Mian Yang 621010, China
liqiangsir@swust.edu.cn

Abstract. Edge-device collaborative computing can provide efficient computing services for emerging intelligent applications by coordinating the resources of terminal devices and edge servers. Given the characteristics of such applications, which are usually composed of multiple dependent subtasks, how to achieve efficient task collaborative processing becomes a key challenge. This paper models dependent tasks as DAG, and its multi-node collaborative scheduling problem has been proven to be an NP-hard problem. To solve this problem, this paper designs a two-stage optimization framework: first, the predicted cost priority is introduced in the task sorting stage, the subtask computational cost is estimated to dynamically adjust the order, and combined with the deep reinforcement learning method, the optimal unloading location is matched for each task based on the proximal policy optimization (PPO) algorithm. Simulation results show that the designed method can effectively reduce the cost of task completion.

Keywords: Edge-device collaboration · Dependency Task · Task Priority · Proximal Policy Optimization (PPO)

1 Introduction

Edge collaborative computing can fully utilize the resources within the system to collaboratively complete tasks and has good performance [1]. At the same time, a number of emerging intelligent service demands have emerged at the edge of the network, such as smart medical care [2] and virtual reality [3]. These service tasks are composed of a series of dependent tasks and rules, and are both latency-sensitive and computationally intensive. Edge-device collaborative use of edge server resources to assist terminal devices can efficiently complete these dependent tasks and reduce system resource consumption [4].

Unlike independent tasks, dependent tasks can effectively shorten the task completion time by executing multiple tasks in parallel. Their dependencies can be modeled as a directed acyclic graph (DAG). DAG can effectively describe the task dependencies, which is conducive to arranging the execution order and location of the task offloading process and realizing accurate resource management. The offloading process of DAG tasks is very complex [5], which has been proved to be a NP-hard problem. Finding a

better offloading solution for edge-device collaborative systems is our main research problem.

To address this problem, articles [6,7] introduced task priorities to determine the optimal execution order of DAG tasks, and a variety of methods were applied to solve the optimal strategy [8,9]. Peng [10] studied the situation where multiple devices interfere with each other and used game theory to coordinate the tasks of multiple users. Wang et al [11] reviewed the advantages and key technologies of deep reinforcement learning for collaborative computing, which shows that the application of deep reinforcement learning methods to task offloading is effective. However, terminal device resources are usually limited, and energy consumption indicators also need to be considered. Therefore, considering the task completion time, energy consumption and interference between devices, the optimization of offloading decisions needs further research. This paper proposes a new priority calculation method that takes into account both latency and energy consumption to obtain a more reasonable task offloading order. It also uses a deep reinforcement learning algorithm to optimize offloading decisions and reduce the system task completion time and energy consumption. The main contributions and innovations of this work are as follows:

1. Different from independent task offloading, this paper studies the collaborative working framework between terminal devices and edge servers, and the offloading scheme can be dynamically adjusted according to task dependencies and cost consumption.
2. Introducing cost priority to sort DAG tasks ensures dependencies and predicts the execution of subsequent tasks at different nodes, which can help allocate tasks. Deep reinforcement learning methods are combined to explore the best offloading solution to achieve a lower system cost.
3. Simulation experiments were conducted on a Python-based simulator to verify the effectiveness of the proposed algorithm in reducing the cost of completing system tasks.

2 System Model

The edge-device collaborative system consists of multiple devices $L = \{L_1, L_2, \dots, L_n\}$ and edge servers $S = \{S_1, S_2, \dots, S_n\}$, and wireless transmission is used for communication between devices. The dependent tasks generated by the devices are described as $DAG = (V_i, E_i)$, where $V_i = \{T_{i,1}, T_{i,2}, \dots\dots, T_{i,n}\}$ represents the subtask collection of the device. Tasks are offloaded through binary and can be executed locally or on the server. If related subtasks are calculated on the same node, communication consumption is not considered. Our goal is to find an optimal offloading solution that minimizes the weighted sum of the time and energy consumption required to complete the computing task.

2.1 Communication and Computational Model

$d_{i,j}$ represents the unloading decision of $T_{i,j}$, the value of 0 represents local execution, and k represents execution on server k .For device i offload decision $d_i = \{d_{i,1}, d_{i,2}, \dots, d_{i,j}\}$, the wireless transmission rate between the device and the server is expressed as:

$$r_{i,k} = B\log_2(1 + \frac{q_i g_{i,k}}{w + \sum_{j \epsilon L: d_i = d_j} p_j g_{j,s}}) \tag{1}$$

Where B, q_i, w represent bandwidth, transmission power and noise power respectively. $\sum_{j \epsilon L: d_i = d_j} p_j g_{j,s}$ quantifies wireless interference from devices linked to the same server, at which time the transmission rate is lower than the theoretical value when the device occupies a channel alone.

The time and energy consumption cost of task $T_{i,j}$ execution on local devices and servers are discussed respectively. The local computing time and energy consumption are expressed as:

$$t_{i,j}^l = \frac{b_{i,j}}{f_l} \tag{2}$$

$$e_{i,j}^l = \delta_i^l b_{i,j} \tag{3}$$

Where $b_{i,j}$ represents the number of CPU cycles required, f_l is the computing power of the device, and δ_i^l is the CPU energy consumption coefficient.

The execution time and energy consumption of the task on the edge server are:

$$t_{i,j}^k = \frac{m_{i,j}}{r_{i,k}} + t_{i,j}^{k,queue} + \frac{b_{i,j}}{f_k} \tag{4}$$

$$e_{i,j}^k = \frac{q_i m_{i,j}}{r_{i,k}} + \sigma t_{i,j}^{k,queue} + \delta_i^k b_{i,j} \tag{5}$$

Where m represents the task size, and the task completion time consists of three parts: transmission time, queuing time, and computing time. The computing energy consumption also comes from these three parts, and σ in the energy consumption expression represents the waiting energy consumption coefficient.

2.2 Problem Formulation

Due to the dependency of DAG tasks, the start and end time of a task is related to the predecessor task (need to wait for the result). In order to accurately calculate the completion time of the task, Start Time (ST) and Finish Time (FT) are defined to represent the start and end time of the task. The calculation method is as follows:

$$ST_{i,j}^k = \max\{avail\{0 \cup [k]\}, \max_{j' \in pred(j)}(FT(i,j') + c_{j',j})\} \tag{6}$$

$$c_{j',j} = \begin{cases} 0, d_{i,j'} = d_{i,j} \\ \frac{data_{j',j}}{r_{i,k}}, other \end{cases} \tag{7}$$

Where $pred(j)$ represents the set of direct predecessor tasks of task j, $c_{j',j}$ is the communication time between subtasks, and $avail\{0 \cup [k]\}$ represents the earliest time when the task can be executed locally or on the server. The end time of task $T_{i,j}$ is expressed as:

$$FT_{i,j}^{k} = ST_{i,j} + RT \tag{8}$$

RT represents the actual execution time. Since the offloading method uses binary offloading, $\alpha_k + \beta_k = 1$ is defined. When the device executes the task locally, $\alpha_k = 1$, and when the edge server executes it, $\beta_k = 1$. Therefore, the task execution time RT and energy consumption are expressed as:

$$RT = \alpha_k t_{i,j}^{l} + \beta_k t_{i,j}^{k} \tag{9}$$

$$RE = \alpha_k e_{i,j}^{l} + \beta_k e_{i,j}^{k} \tag{10}$$

Our goal is to minimize the system task completion cost, and the cost is defined as the weighted sum of time and energy consumption. The optimization problem is expressed as:

$$min\, COST = \omega^t \sum_{i=1}^{N} RT_i + \omega^e \sum_{i=1}^{N} RE_i \tag{11}$$

Where ω^t and ω^e represent the weight ratios of time and energy consumption of different unloading decisions.

2.3 Task Priority

When sorting tasks, we first need to ensure dependencies and further explore a more reasonable task execution order to help optimize the offloading strategy. In this paper, we introduce a priority with forward-looking characteristics, which not only considers the cost of the current execution task, but also considers the importance of the dependent predecessor and successor tasks. We call it the predicted cost priority, which is expressed as:

$$Pri(T_{i,j}, m) = \begin{cases} COST(j, m), j = j_{exit} \\ COST(j, m) + \max\limits_{j' \in succ(j)} \left[\min\limits_{n \in S} (Pri(T_{i,j'}, n) + COST(j', n)) \right] \end{cases} \tag{12}$$

For exit tasks, the priority value is the completion cost of task $T_{i,j}$. For non-exit tasks, not only the current task cost is calculated but also the minimum cost value that can be obtained by the subsequent task $T_{i,j'}$. $Pri(T_{i,j}, m)$ is the sum of the execution cost of task $T_{i,j}$ and the longest critical path cost of the successor task. It is calculated through reverse dynamic programming to ensure that the priority value of each task depends on

the optimal solution of its successor task. The maximum value operation reflects the maximum path cost, which is in line with the critical path principle in task scheduling.

3 Algorithm Formula

3.1 Modeling the Offloading Process as MDP

The edge-device collaborative task offloading process is expressed as MDP, and the optimal offloading strategy is obtained through the continuous interaction between the agent and the environment. MDP is usually represented in the form of a five-tuple, $M = < S, A, R, P, \gamma >$. The state, action, and reward value of each time slot t are s_t, a_t, r_t, P represents the state transition probability, and $\gamma \in [0,1]$ is the discount factor. The specific definitions of state, action, and reward are as follows.

State: The state of a time slot t can be $S = \{W_t, C_t, pred_t, succ_t\}$, where W_t is the time set of the task, C_t represents the cost of the task, $pred_t$ represents the predecessor task set of the DAG task, and $succ_t$ represents the successor task set of each DAG task.

Action: The interaction process between the agent and the environment requires continuous adjustment of actions to obtain the optimal offloading strategy. The action in this article is defined as the selection of the offloading location, that is, local computing or offloading to the server computing.

Reward: For each action performed, a corresponding reward is generated to judge whether the action is moving in the direction of the goal. This paper associates the goal of minimizing the cost of completing the task with the reward, so the instant reward is defined as the cost $COST$ defined by (11).

3.2 Priority Offloading Algorithm Based on PPO

In order to obtain the optimal unloading solution, we used the DRL method based on PPO for policy optimization. The PPO algorithm is a policy-based reinforcement learning algorithm with high sample efficiency. We define the offloading decision of the device at time t as $\pi(a_t|s_t)$, which represents the probability of taking action a_t in state s_t under the current strategy. The optimal offloading strategy is:

$$a_t = arg \max_{a_t} \pi(a_t|s_t) \tag{13}$$

The actor is combined with the critic, and the critic network is used to continuously adjust the actor to obtain the optimal action. In reference [12], θ is defined as the actor network parameter, and the following formula is used to represent the policy loss function for updating the actor network.

$$L^{CLIP}(\theta) = \mathbb{E}_t[min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{14}$$

Where

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{15}$$

$r_t(\theta)$ is the ratio of the new policy π_θ to the old policy $\pi_{\theta_{\text{old}}}$, which indicates the update amplitude of the policy. clip$(r_t(\theta),1-\epsilon,1+\epsilon)$ is a clipping function that limits $r_t(\theta)$ to the interval $[1-\epsilon,1+\epsilon]$. $\hat{A}_t$ is the advantage function.

Define ϕ as the Critic network parameter, and we update the loss function using the following formula:

$$L(\phi)=\frac{1}{2}\mathbb{E}_t[(V_\phi(s_t)-V_{target})^2] \tag{16}$$

Where

$$V_{\text{target}}=\sum_{i=t}^{T-1}\gamma^{i-t}r_i+\gamma^{T-t}V_\phi(s_T) \tag{17}$$

γ is the discount factor, and the immediate reward at time step i is denoted as r_i.

The cost priority based PPO algorithm (COPPO) is shown in Algorithm 1. In steps 1-3 of the algorithm, the state and network parameters are initialized first. In steps 4-6, the tasks are updated according to priority, and parameters such as reward values are collected through multiple trainings. Finally, the advantage estimate is calculated and the network parameters are updated.

Algorithm 1 COPPO

1: Initialize state s_0
2: Initialize policy π_{θ_0}
3: Initialize parameters ϕ_0 and θ_0
4: Calculate the priority factor $Pri(T_{i,j})$ for each task
5: Sort tasks and update task list
6: **repeat**
7: Run policy $\pi_{\theta_{\text{old}}}$ for T steps, collect the$\{s_t,a_t,r_t\}$
8: Compute the advantage estimates $\hat{A}_t,\ldots,\hat{A}_T$
9: Update θ and ϕ with M epochs by the gradient descent method
10: Update θ_{old} with θ
11: **until** $episode > episode_{max}$

4 Simulation Results and Analysis

To verify the effectiveness of the designed algorithm in reducing costs, simulation experiments are conducted and the results are compared with other solutions. The system environment mainly consists of devices, edge servers, and wireless channels. We generate DAG tasks for simulation experiments. The main parameter settings of the experiment are shown in Table 1.

Table 1. Simulation parameters

Parameter	Value
Device computing capacity f_l	500MHz
Server computing capacity f_k	5GHz
Subtask size $m_{i,j}$	1-5MB
Subtask required CPU cycles $b_{i,j}$	20-1000 Megacycles
Transmission power q_i	100mW
Noise power w	-100dBm
Discount factor γ	0.95
Clip Parameter ϵ	0.2
Learning rate	1e-4
Size of a mini-batch	256
Hidden layer size	(512,512)
Cost weight ω^t, ω^e	0.5,0.5

4.1 Algorithm Convergence

Fig. 1. Convergence curve Fig. 1 shows how the task completion cost changes with the number of training rounds. As shown in the figure, we set the experimental conditions with 5 edge servers, 40 devices, 10 edge servers and 50 devices.

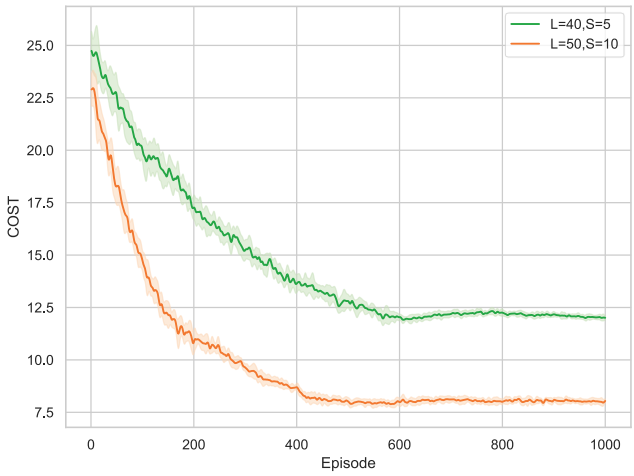


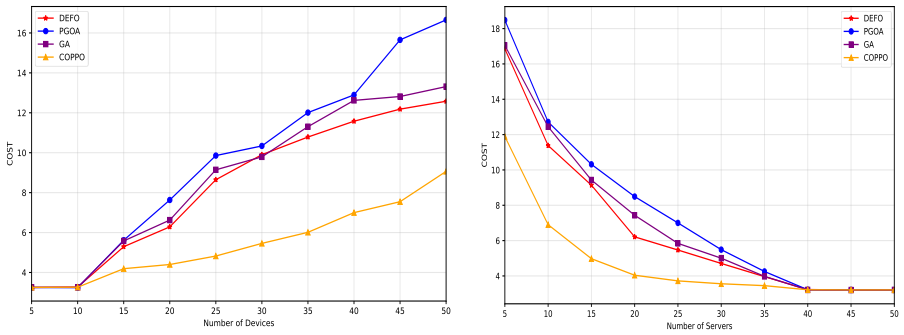
Fig. 1. Convergence curve

As the number of iterations increases in Fig. 1, the cost values under both conditions show a downward trend and tend to stabilize after a finite number of trainings. The 95% confidence interval band in the figure shows the fluctuation range of the cost value

during the training process, demonstrating the stability of the algorithm in this paper. In summary, it can be concluded that the algorithm proposed in this paper is convergent.

4.2 Algorithm Comparison

Our comparison methods are the potential game theory based task scheduling algorithm (PGOA) [13], the sustainable task offloading based on genetic algorithm (GA) [14]and. the distributed fine-grained task offloading algorithm (DEFO) [15].



(a)Change device number (b) Change server number

Fig. 2. Effects of different conditions

Fig. 2(a) shows the impact of the terminal device scale on the cost when 10 edge servers are fixed. As the number of devices increases from 5 to 50, the system cost of all algorithms shows an approximately linear growth trend. In the low-load stage (N$\leqslant$ 10), each device can monopolize server resources and the system maintains the minimum cost state; when N>10, resource competition leads to increased processing delay and energy consumption. In Fig. 2(b), the number of terminal devices in the edge computing network is set to 40, and then the number of edge servers increases from 5 to 50. As the number of edge servers in the system increases, the system cost tends to decrease. This is because the more edge servers there are, the more sufficient computing resources there are in the system, and tasks can be completed in a timely manner, reducing the corresponding processing time and energy consumption. However, when the number of edge servers increases to 40, the cost of completing tasks no longer decreases, and each device has an edge computing server to assist.

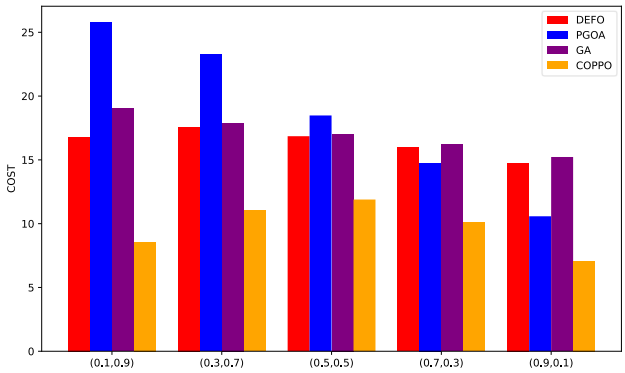


Fig. 3. Effects of different cost weights

Fig. 3 shows the impact of different time and energy consumption weights on the cost of task completion. It can be seen from the figure that COPPO achieves the lowest cost consumption among all algorithms. And under different weight settings, the costs of various algorithms are also different. This is because the algorithms have different focuses, which in turn affects the offloading decision.

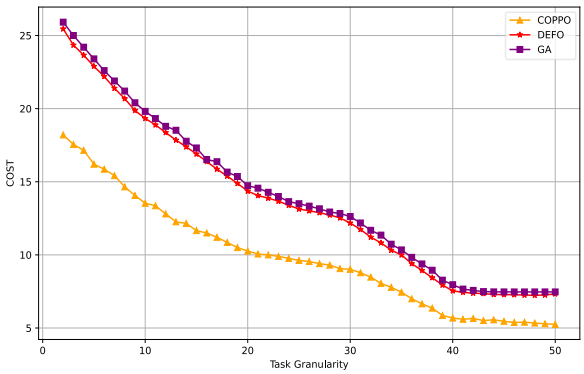


Fig. 4. Effect of different task granularity on cost

In order to further study the impact of changes in DAG task attributes on the cost of task completion, we simulated the impact of increasing the number of subtasks of a task and dividing the task in a more fine-grained manner on the cost. It is worth noting that the resources required for the task remain unchanged. The results in Fig. 4 show that the cost of completing the task is effectively reduced in this way, because the finer-grained division can further improve the resource utilization efficiency of the edge server.

Considering the two-stage optimization structure of the algorithm, the time complexity of task sorting is $O(l_n(e_n + v_n))$, which is mainly related to the size of the DAG task. The deep reinforcement learning method will explore and find the best action

strategy, and the time complexity is $O(2^n)$, so the total algorithm time complexity is $O(2^n)$.

5 Conclusions

This paper designs a dependent task offloading algorithm to study the task offloading under the edge-device collaborative architecture to reduce the task completion cost of the edge-device collaborative system. We define the dependent tasks of the device as DAG. The predicted cost priority is introduced to generate the task offloading list, which comprehensively considers the completion time and energy consumption of the task. Then the PPO algorithm is used for task offloading optimization training to achieve task offloading at the lowest cost. The simulation experimental results show that compared with other similar works, the algorithm in this paper maintains a lower cost, which verifies the effectiveness of the proposed algorithm.

References

1. Chen, H., Qin, W., Wang, L.: Task partitioning and offloading in IoT cloud-edge collaborative computing framework: a survey. J. Cloud Comput. **11**(1), 86 (2022)
2. Yang, Z., Liang, B., Ji, W.: An intelligent end–edge–cloud architecture for visual IoT-assisted healthcare systems. IEEE Internet Things J. **8**(23), 16779–16786 (2021)
3. Chen, Z., Zhu, H., Song, L., Xu, X., Zhang, Z.: Wireless multiplayer interactive virtual reality game systems with edge computing: modeling and optimization. IEEE Trans. Wireless Commun. **21**(11), 9684–9699 (2022)
4. Jeremiah, S.R., Yang, L.T., Park, J.H.: Digital twin-assisted resource allocation framework based on edge collaboration for vehicular edge computing. Future Gener. Comp. Sy. **150**, 243–254 (2024)
5. Gonçalves, G.E., Endo, P.T., Rodrigues, M., Sadok, D.H., Kelner, J., Curescu, C.: Resource allocation based on redundancy models for high availability cloud. Comput. **102**(1), 43–63 (2020)
6. Liu, S., Yu, Y., Lian, X., Feng, Y., She, C., Yeoh, P.L., Guo, L., Vucetic, B., Li, Y.: Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks. IEEE J Sel Areas Commun. **41**(2), 538–554 (2023)
7. Djigal, H., Feng, J., Lu, J.: Task scheduling for heterogeneous computing using a predict cost matrix. In: Workshop Proc. 48th Int. Conf. Parallel Process, pp. 1–10 (2019)
8. Chai, F., Zhang, Q., Yao, H., Su, Z., Qin, Y., Guo, S.: Joint multi-task offloading and resource allocation for mobile edge computing systems in satellite IoT. IEEE Trans. Veh. Technol. **72**(6), 7783–7795 (2023)
9. Wang, Z., Goudarzi, M., Gong, M., Wu, H., Yu, H.: Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog computing environments. Future Gener. Comp. Sy. **152**, 55–69 (2024)
10. Peng, B., Chen, C., Li, Q., Zhang, T., Liu, Y.: Latency-optimized multi-user task offloading scheme using dynamic priority and duplication in edge computing. In: Proc. 29th IEEE Int. Conf. Parallel Distrib. Syst. (ICPADS), pp. 1093–1098 (2023)

11. Wang, Y., Yang, C., Lan, S., Zhu, L., Zhang, Y.: End–edge–cloud collaborative computing for deep learning: a comprehensive survey. IEEE Commun. Surv Tut. **26**(4), 2647–2683 (2024).
12. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
13. Yang, L., Zhang, H., Li, X., Chen, M., Leung, V.C.M.: A distributed computation offloading strategy in small-cell networks integrated with mobile edge computing. TON. **26**(6), 2762–2773 (2018)
14. Shu, C., Zhao, Z., Han, Y., Min, G., Duan, H.: Multi-user offloading for edge computing networks: a dependency-aware and latency-optimal approach. IEEE Internet Things J. **7**(3), 1678–1689 (2019)
15. Chakraborty, S., Mazumdar, K.: Sustainable task offloading decision using genetic algorithm in sensor mobile edge computing. J. King Saud Univ. Comput. **34**(4), 1552–1568 (2022)