Edge Computing Energy-Efficient Resource Scheduling Based on Deep Reinforcement Learning and Imitation Learning

Hengliang Tang[1], Rongxin Jiao[1(✉)], Tingting Dong[2], Huilin Qin[1], and Fei Xue[1]

[1] School of Information, Beijing Wuzi University, Beijing 101149, China
jeorgea@163.com

[2] Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China

Abstract. Task scheduling is one of the key technologies in edge computing. End devices can significantly improve the quality of service by offloading some latency-sensitive tasks to edge servers, but a large amount of power and compute units are wasted. Therefore, this paper proposes a two-stage task offloading approach to ensure low latency while reducing the energy consumption of edge computing units and cloud computing centers. The mobile edge computing environment contains edge computing nodes as well as cloud computing centers. A two-stage processing mechanism based on deep Q-learning is used to automatically generate optimal long-term scheduling decisions that reduce power consumption while ensuring quality of service. Imitation learning is also used in the reinforcement learning process to reduce the training time of the optimal policy. To evaluate the effectiveness of the model, we use the Shortest job first (SJF) algorithm and the Heterogeneous Earliest Finish Time (HEFT) into as comparison algorithms, comparing the running time and energy consumption as a measure. Our proposed algorithm has 13% more running time but 34% lower average energy consumption compared to other algorithms.

Keywords: Edge computing · Resource scheduling · Deep reinforcement learning · Imitation learning

1 Introduction

In the past few years, with the growth of data volume of terminal devices and the continuous development of artificial intelligence applications, the limited computing power of terminal devices cannot meet the application scenarios such as unmanned driving, smart manufacturing and smart cities. This has led to the proposed edge computing service model. Edge computing is the processing of latency-sensitive partial requests through micro computing units such as base stations in the vicinity of mobile end devices, thus improving the response speed as well as the stability and continuity of services. Requests for non-delay-sensitive types are processed by means of cloud computing, thus reducing the load pressure on the edge computing units. Currently, the edge processing units are underutilized, leading to idle equipment and energy waste. The data shows [1]. For

L. Pan et al. (Eds.): BIC-TA 2021, CCIS 1566, pp. 222–231, 2022.
https://doi.org/10.1007/978-981-19-1253-5_16

traditional computing centers, the optimal energy utilization is 70–80%, but the actual service delivery is unpredictable in terms of user requests, so the energy utilization is usually below 50%. We need to simulate user requests and find a reasonable scheduling scheme to ensure the quality of service and energy consumption.

In the MEC (Mobile Edge Computing) scenario, by deploying micro-servers in the communication base stations near the terminal devices, part of the computing resources are sunk to the user side, which shortens the distance on the physical path, reduces the communication overhead, and also relieves the load pressure on the cloud computing center. The whole edge computing service system has two service parties, which are the edge computing nodes and the cloud computing center. The service process of the system starts with the end device sending requests to the edge computing node, and the edge computing node sends part of the non-delay-sensitive requests to the cloud for processing back to the edge server, and then the edge server returns to the end device. For some delay-sensitive requests, the edge computing nodes process and return the results directly to the terminal.

Traditional deep learning resource management schemes have long training time and slow convergence [2]. This is mainly due to the fact that RL is completely ignorant of "expert knowledge", which allows RL to learn the optimal policy from scratch through trial and error, which motivates us to adopt a more efficient training method [3]. The training time can be significantly reduced by imitation learning.

In this project, the system consists of two main parts: workload processor and two-level resource decision system. The workload processor receives user requests and adds them to the task queue of the edge computing nodes and the task queue of the cloud computing center, respectively. The two-level resource decision system allocates the appropriate computing units based on the provided task information. To improve the processing efficiency of compute units, we dynamically update the ready task queues to achieve the parallel operation of the impractical virtual machines (VMS). Our task scheduling optimization goal is to minimize the long-term energy cost by training while ensuring the quality of service. We first implement this system and then verify the effectiveness of our system by benchmarking algorithms. Our main contributions are listed below.

(1) We implemented a two-level scheduling algorithm based on deep Q-learning. The two-level decision can minimize the energy cost while satisfying the service quality.
(2) We built an edge computing service simulation system to simulate the requests of end devices, edge computing service devices, resources and other information.
(3) To speed up model training and convergence, we integrate imitation learning in reinforcement learning to mimic the behavior of some classical heuristics (e.g., minimum average waiting time). These scheduling methods are considered as 'experts' for specific scenarios, thus improving the effectiveness of agent training and substantially reducing the training time.

2 Related Works

In recent years, academic circles have been enthusiastic about research in the field of MEC: at the theoretical level, a number of literature reviews [4] summarized the

existing theoretical framework and architecture of MEC and looked forward to the future development of MEC; Mao Y et al. [5] investigated MEC supporting technologies, including technologies such as virtualization and software-defined networking; Zhang K et al. [6] classified MEC scenes, models, and deployment in detail. At the application level, Yang X et al. [7] expounded the application and value of MEC in the field of Internet of Things; Huang X [8] et al.

Computing offloading is a process in which mobile users delegate computing-intensive tasks to cloud computing platforms for execution. It has been fully and widely applied in the field of MCC [9]. In the field of MEC, computing offloading technology is the first of the three design elements of the MEC system (the other two major elements are resource allocation and mobility management), which can not only optimize resource utilization, but also reduce service delay, extend equipment life, and improve user experience, so it is also of great research value [10]. In general, the key to the design of the calculation offloading algorithm is to decide which tasks to uninstall and which devices to offload tasks to execute. Starting from the scope of computational offloading, Yang J et al. [11] divided the computational offloading model into two basic categories: single server model and multi-server model.

Under the single-server computing offloading model, Yuan J et al. [12] used a convex optimization method to achieve the optimal computational offloading decision.

At present, there is little research on edge computing resource allocation using deep reinforcement learning algorithm. On the basis of [1], combined with Deep Reinforcement Learning (DQN) algorithm and edge computing environment model, we propose an edge computing task allocation model for energy efficient utilization, and use imitation learning method to speed up the model training speed and realize the efficient use of energy under the condition of meeting task constraints.

3 Scheduling System

We expect our system to minimize long-term energy consumption. To achieve this goal, we built a platform that simulates the MEC environment model, which calculates energy consumption and costs. We use real user request data and the workload model is handled by the user. After that, we built an environmental model that provides information on resource usage and energy consumption.

3.1 Workload Processor

Our system contains edge computing node processing units as well as cloud computing center virtual machine processing units, where the edge node device set is represented by set E_i and the computing center virtual machine is represented by set VM_i. The set of devices is represented as Eqs. 1 and 2.

$$E = \{E_1, E_2, \cdots, E_n\} \tag{1}$$

$$VM = \{VM_1, VM_2, \cdots, VM_n\} \tag{2}$$

We abstract each user request into a DAG graph, which is represented by a binary G < V, E >, where V is the task vertex and E is the edge associated with the vertex. Each vertex represents a task, and the edge represents the relationship between the two tasks. The workload processor is responsible for classifying the tasks in the DAG diagram, and then dynamically adding sub tasks to the edge node task queue Q_e, T in the set of Q_e denotes the set of task queues on edge node e and the cloud environment task queue Q_c, T in the set denotes the set of task queues on the central server. The above two sets are expressed by the following Eqs. 3 and 4.

$$Q_e = \left\{ T_1^E, T_2^E, \cdots, T_n^E \right\} \tag{3}$$

$$Q_c = \left\{ T_1^C, T_2^C, \cdots, T_n^C \right\} \tag{4}$$

Each task is represented by Ti. The workload resolves Ti based on the information provided by the dataset include Ti_{start} (Start execution time), Ti_{ddl} (Task deadline), Ti_{res} (Computing resources required), Ti_s (Whether the task is delay-sensitive is represented by a binary number, where 1 is delay-sensitive) and relevance to other tasks. Relevance refers to the constraint relationship before and after task execution. For example, if task a depends on the results and data of Task B, task a can only be executed after task B is completed. Tasks are added to different queues according to the Tis value, and the process is as in Eq. 5.

$$\begin{cases} Add\ Q_e(T),\ if\ T_{tag} = 1 \\ Add\ Q_c(T),\ if\ T_{tag} = 0 \end{cases} \tag{5}$$

T_i^{time} indicates that the workload processor calculates the running time required for each task at different nodes, where i represents the task number. The runtime on edge nodes and compute center VMs is calculated by the following Eqs. 5 and 6.

$$T_i^{time} = \frac{T_i res}{VM_i^p} \tag{6}$$

$$T_i^{time} = \frac{T_i res}{E_i^p} \tag{7}$$

We use the HEFT algorithm to obtain the priority queue Q to provide to the agent for selecting the appropriate computational unit. The HEFT algorithm is a static scheduling algorithm that calculates the priority of tasks by computing the rank value from the bottom up. It first calculates the earliest completion time of all tasks on each virtual machine according to the dependencies in the DAG before scheduling, then compares the smallest earliest completion time and its corresponding computational unit, matches the task with the virtual machine and corresponds to the corresponding occupied time slot, and then schedules the tasks of the whole workflow to the corresponding virtual machine after all tasks and virtual machines are matched. After all the tasks and virtual machines are matched, the tasks of the entire workflow are scheduled to the corresponding virtual machines for operation.

3.2 Problem Definition

We use deep reinforcement learning methods to generate reasonable task allocation schemes that minimize energy consumption while meeting latency requirements and quality of service. Therefore, we build a 2-level decision system. to handle task queues at the edge nodes as well as at the cloud computing center, respectively. Figure 1 shows the flowchart of task queue processing by the smart body. The task queues are generated by the Workload Processor and used as the input to the neural network. The intelligence automatically generates the best decision based on the reward function.

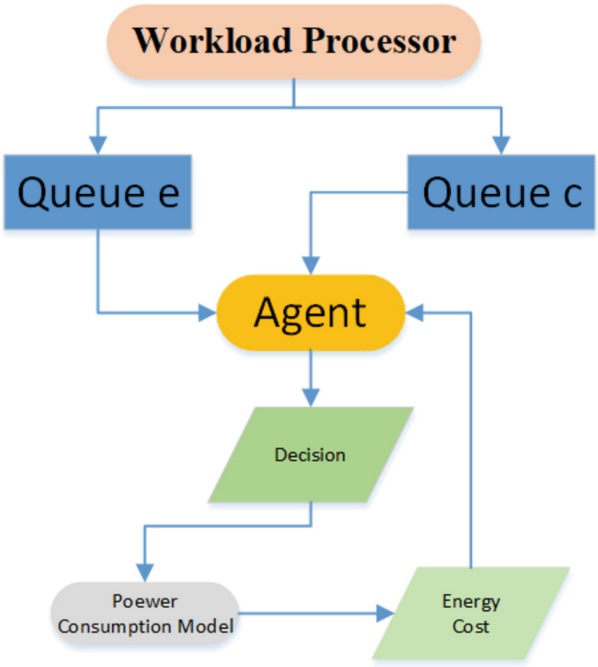

Fig. 1. Flow chart of system.

Power Consumption Model: We calculate the utilization of node i by expressing it in the following equation, where Runtime(m) is the total running time of the node, $\sum_{i=1}^{n} \mathrm{Ti}_{res}^{time}$ is the amount of time being occupied, where Ti_{res}^{time} is the amount of resources required for each task, and i represents the tasks that currently running on this node.

$$U_m = \frac{\sum_{i=1}^{n} T_i^{time}}{Runtime} \tag{8}$$

The power of a computational node consists of two components: static power Pwrs and dynamic power Pwrd. where the static power is fixed while the node is operating,

while the dynamic power varies as the utilization increases.

$$Pwr_s = \begin{cases} 0 & U_m = 0 \\ const & U_m > 0 \end{cases} \tag{9}$$

$$Pwr_D = \begin{cases} \alpha * U_m & U_m < \tilde{U}_m \\ \alpha * \tilde{U}_m + (U_m - 0.7)^2 \beta & U_m \geq \tilde{U}_m \end{cases} \tag{10}$$

Um is the utilization rate of server m, $\alpha = 0.5$, $\beta = 10$, $\tilde{U}_m$ is the optimal utilization which is 0.7 [13]. And the total power of server m Pwrm is the sum of static power Pwrs and dynamic power Pwrd.

Each task's runtime takes up a non-infinite amount of computational resources. In reality, it will be automatically released at the end of the process. To simulate this process, we set a random runtime for each task. During this time, some resources of the computational unit will be occupied by that task. In other words, when the agent assigns a task to a computing unit, we can get an end time which represents the expected completion time of the task. When the occupancy time is over, we change the status of the task from "ready" to "finished" and release the corresponding resources. In this way, we can dynamically model the changes of resources in the server system and help the agent to make decisions.

3.3 Environment Model

Figure 2 below shows the workflow of the two-level decision processor [14, 15]. Tasks are added to the edge node compute waiting queue as well as the cloud computing center waiting queue based on latency sensitive markers, respectively. The system keeps track of the resource status, information, records of the current cloud server platform and basic information about the new tasks. The agent makes a decision based on the current task and status information through a reward function. The pseudo code of two-layer deep reinforcement learning is shown in Fig. 3.

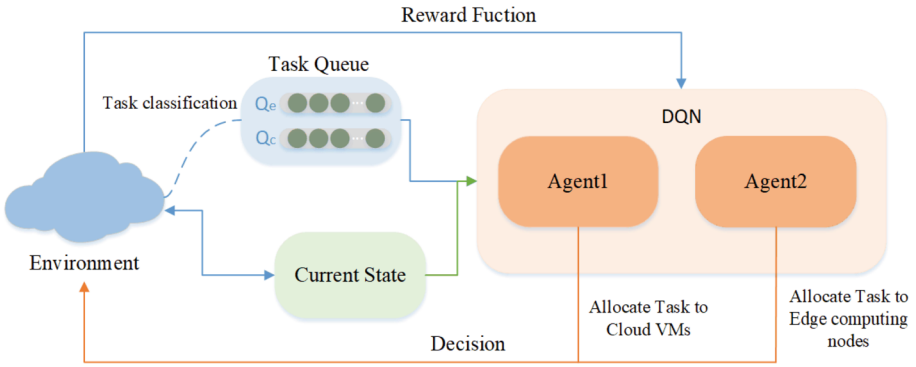

Fig. 2. Two-stage processor based on deep reinforcement learning.

State Space: In the two-stage decision-making process, the current state information and task information are obtained as the input information of the environment. For

the agent of stage1, it will get the current resource status information and single task information. For the second stage, the agent will obtain the resource status information of all base stations and the task information of virtual machines in the base station.

Action Space: Just like the state space, the action space defines the operations available to the agent [16]. In the first stage, DQN needs to select a server farm from all the server farms of the current task, and the agent of stage2 will select the appropriate base station from the base station set.

Reward Functions: The following are the reward functions of our two agents [15].

$$R = P\left[t_n, Pwr_D(t_{n-1}) - Pwr_D(t_n)\right] \tag{11}$$

t_n is the current time, and $Pwr_f(t)$ represents the total power of servers in the farm on time t, and $Pwr_m(t)$ represents the power of individual server on time t.

DQN Resource Scheduling Algorithm

Input: Ready Tasks

Output: Scheduling Matrix

01. Initialize environment information
02. Initialize workload processor
03. $Q_e \leftarrow$ workloadprocessor() , $Q_c \leftarrow$ workloadprocessor() //Get Task Queue
04. **For** each Calculation unit do C_i
05. **For** each task unit do T_i
06. Unit_Task_ arrays $\leftarrow$ calculate T_i^{time} //Calculate the T_i runtime on C_i
07. **For** each task in Qe
08. Scheduling Matrix $\leftarrow$ RunAgent1(Unit_Task_ arrays)
09. **For** each task in Qc
10. Scheduling Matrix $\leftarrow$ RunAgent2(Unit_Task_ arrays)
11. Return Scheduling Matrix

Fig. 3. Resource scheduling algorithm pseudo code.

4 Simulation Experiment

To verify the effectiveness of our algorithm, we compared the proposed algorithm HEFT* with HEFT and Shortest Job First (SJF) algorithms. We used a real Google server log dataset for the experimental comparison. Experimental environment parameters set 100 edge computing nodes and 3 cloud computing centers, We use imitation learning to improve the convergence efficiency of the model.

The experimental environment is a computer with Windows 10 Home Edition, Intel i7–6700 CPU, 16G memory, and NVIDIA GeForce GTX 1050Ti graphics card with 8G memory.

We use the above parameters and data set and the experimental results are as follows (Figs. 4 and 5).

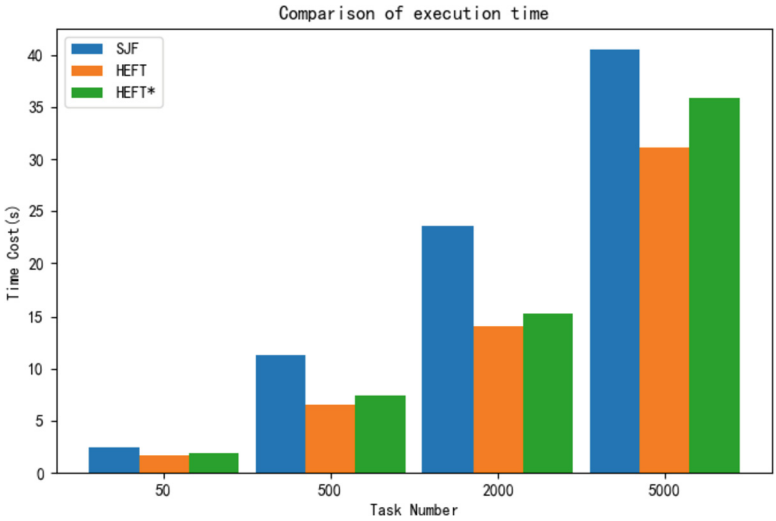

Fig. 4. Scheduling runtime comparison.

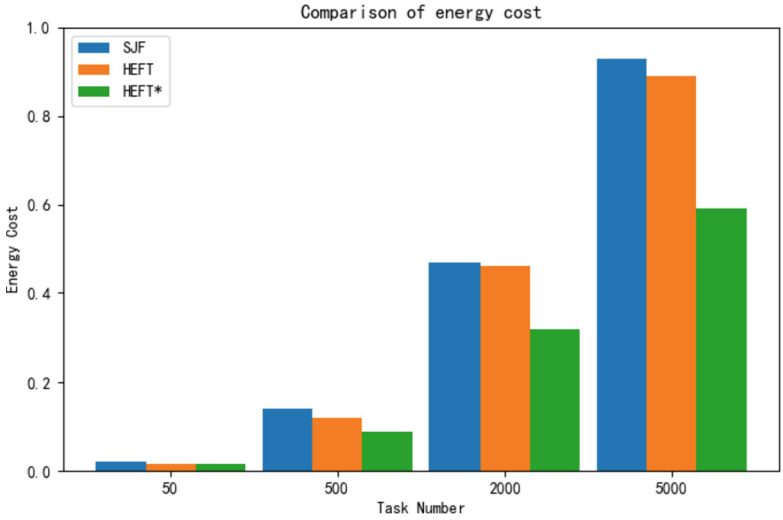

Fig. 5. Energy cost comparison.

This shows that the execution time of our proposed algorithm HEFT* is on average 13% more compared to the HEFT algorithm, but the average energy consumption of scheduling is reduced by 34%. Because the DQN agent learns from the reward function and continuously selects the server group and base station with the largest reward.

5 Conclusion

The task scheduling problem in edge computing environments requires rational algorithms to reduce energy consumption. Although traditional algorithms have shorter scheduling time, their energy consumption is not considered.

The HEFT* algorithm is experimentally tested in an environment with edge computing nodes and cloud computing virtual machines. The time required for task scheduling and the energy consumption are tested. And the convergence time is reduced by using imitation learning method in training the DQN model. Task fault tolerance, network failure, etc. can be considered.

In the future, a combination of fault-tolerant approaches to task fault tolerance in edge computing environments such as those considered in [17–19] and more precise methods for algorithms to determine whether a task is latency-sensitive can be considered to be closer to the real situation.

References

1. Kozyrakis, C.: Resource efficient computing for warehouse-scale datacenters. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1351–1356 (2013)
2. Tong, Z., Chen, H., Deng, X., Li, K., Li, K.: A scheduling scheme in the cloud computing environment using deep Q-learning. Inf. Sci. **512**, 1170–1191 (2020)
3. Wen, G., et al.: Cloud resource scheduling with deep reinforcement learning and imitation learning. IEEE Internet Things J. **8**(5), 3576–3586 (2021)
4. Bellavista, P., Chessa, S., Foschini, L., et al.: Human-enabled edge computing: exploiting the crowd as a dynamic extension of mobile edge computing. IEEE Commun. **56**(1), 145–155 (2018)
5. Mao, Y., Zhang, J., Song, S.H., et al.: Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems. IEEE Trans. Wireless Commun. **16**(9), 5994–6009 (2017)
6. Zhang, K., Mao, Y., Leng, S., et al.: Mobile-edge computing for vehicular networks: a promising network paradigm with predictive off-loading. IEEE Veh. Technol. Mag. **12**(2), 36–44 (2017)
7. Yang, X., Chen, Z., Li, K., et al.: Communication-constrained mobile edge computing systems for wireless virtual reality: scheduling and trade-off. IEEE Access **6**, 16665–16677 (2018)
8. Huang, X., Yu, R., Kang, J., et al.: Distributed reputation management for secure and efficient vehicular edge computing and networks. IEEE Access **1**, 99 (2017)
9. Xu, X., Liu, J., Tao, X.: Mobile edge computing enhanced adaptive bitrate video delivery with joint cache and radio resource allocation. IEEE Access **1**, 99 (2017)
10. Morabito, R., Cozzolino, V., Ding, A.Y., et al.: Consolidate IoT edge computing with lightweight virtualization. IEEE Netw. **32**(1), 102–111 (2018)
11. Duan, Y., Sun, X., Che, H., et al.: Modeling data, information and knowledge for security protection of hybrid IoT and edge resources. IEEE Access **7**, 1 (2019)
12. Liu, Z., Choo, K.K.R., Grossschadl, J.: Securing edge devices in the post-quantum Internet of Things using lattice-based cryptography. IEEE Commun. Mag. **56**(2), 158–162 (2018)
13. Yang, J., Zhihui, L., Jie, W.: Smart-toy-edge-computing-oriented data exchange based on blockchain. J. Syst. Archit. **87**, 36–48 (2018). https://doi.org/10.1016/j.sysarc.2018.05.001
14. Gao, Y., et al.: An energy and deadline aware resource provisioning, scheduling and optimization framework for cloud systems. In: Hardware/Software Co-Design and System Synthesis. IEEE, pp. 1–10 (2013)

15. Mao, H., Alizadeh, M., Menache, I., Kandula, S.: Resource management with deep reinforcement learning. In: Proceedings of the 15th ACM Workshop on Hot Topics In Networks - HotNets 2016 (2016). https://doi.org/10.1145/3005745.3005750
16. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature, **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236
17. Xie, G., Zeng, G., Li, R., et al.: Quantitative fault-tolerance for reliable workflows on heterogeneous IaaS clouds. IEEE Trans. Cloud Comput. **8**(4), 1223–1236 (2020)
18. Moon, J., Jeong, J.: Smart manufacturing scheduling system: DQN based on cooperative edge computing. IMCOM, 1–8 (2021)
19. Wu, Y., Dinh, T., Fu, Y., et al.: A hybrid DQN and optimization approach for strategy and resource allocation in MEC networks. IEEE Trans. Wirel. Commun. **20**(7), 4282–4295 (2021)