Energy Consumption Optimization of Edge Computing Based on Reinforcement Learning

XiangYu Jiang
Department of electronic and electrical engineering
Taizhou Institute of Sci.&Tech., NJUST.
Taizhou, Jiangsu, 225306, China
jxylingyue@163.com

Abstract. Cloud computing faces the problem of over-centralization, where the distance between users and cloud servers is very far. The communication between users and clouds faces long communication latency and high computational energy consumption. Edge computing is a hot research topic in academia now. Edge computing sinks cloud computing nodes to the edge, allowing users to offload tasks directly to edge servers. Compared with cloud computing, edge computing is closer to the user side. The communication between users and edge servers has lower transmission latency and less energy consumption. In order to better exploit the features and promote the development of edge computing, in this paper, we mainly focus on exploring the basic architecture of edge computing. In addition, we propose a scheme that can optimize the energy consumption of edge computing based on reinforcement learning methods. Finally, we verify the effectiveness of our proposed scheme by comparing it with other schemes through simulation experiments.

1. Introduction

1.1. The background of edge computing
The number of IoT devices is increasing dramatically worldwide, which is expected to reach approximately 24.6 billion connected devices in 2025 [1]. This is a very impressive figure. We can learn that the significant increase of these devices is a pretty strict challenge for data processing capabilities. In the past, we usually used cloud computing solutions to upload any data, information, tasks, etc. to cloud servers and wait for cloud servers to process our requests. We had to face long waiting times, intolerable connection stability, and frequent privacy and security breaches. An interesting idea in academia now is to bring servers down to the edge, that is, closer to the user. We usually call it edge computing. It allows users to offload services that are difficult to process locally to a nearby edge server for processing. Due to the shortened transmission distance, the communication latency between the user and the server is drastically reduced. So it can meet the requirement of low latency for IoT devices. In addition, edge computing keeps the data around the user and avoids transmission to centralized servers, and the user's data security is guaranteed [2].

1.2. High energy consumption of 5G
The advent of 5th generation mobile networks (5G) brings higher rates, greater bandwidth, and lower latency to data communication [3]. And 5G services are based on a service-oriented architecture, so it
could invoke service requirements flexibly and allow us to have flexible and dynamic access to edge servers. We can expect that in the future, various types of edge computing will be directly integrated in the 5G network architecture [4]. This will bring a broader prospect for edge computing.

However, the truth is not always as rosy as we imagine. Due to the high-frequency nature of 5G, the energy consumption caused by the data during transmission is extremely high. On top of that, the user's tasks are offloaded to the edge server, and the computation and processing of data in the edge server cause another part of energy consumption. The huge energy consumption means high cost, which will make edge computing exist only in thesis and difficult to spread to all industries. According to [5], global CO2 emissions have reached 34041045974 tons in 2018. In order to save energy and reduce costs, we must find a feasible way to optimize the energy consumption of edge computing.

1.3. Contributions
In this paper, our contributions are mainly as follows: (1) We build a four-tier edge computing architecture to facilitate computation offloading by users. (2) We propose an Energy Consumption Optimization Algorithm (ECOA) based on the Q-learning algorithm to reduce the energy consumption caused by edge computing. (3) The effectiveness of the method is verified by comparing it with several algorithms. The remainder of the paper is organized as follows: In section 2, we briefly introduce the related work. In section 3, we describe the architecture of edge computing. Section 4 introduces the energy consumption model. Section 5 shows the Markov decision process and Q-learning-based ECOA. And then, the effectiveness of the method is proved by simulation experiments in 6. Finally, section 7 obtains the conclusion and summarizes the paper.

2. Related work

2.1. Edge computing architectures
A number of scholars have proposed edge computing architectures, which are usually presented as layered architectures that are easier to separate and decouple each layer. For example, Li et al. describe a simple two-layer edge computing architecture. The bottom layer is the User Equipment (UE), consisting of a variety of mobile devices, which are responsible for data collection [6]. The upper layer is the edge computing layer, which consists of a large number of edge servers that are responsible for task processing and computation. In other words, the UEs offload their tasks to the edge servers for computing. In addition, an alternative edge computing architecture design scenario is described by Abbas et al. A cloud data center layer is added to the two-tier architecture, which is responsible for persistently storing the data in the edge computing layer [7]. These already existing edge computing architectures can complete the work of edge computing, but not enough. For example, the detailed scheduling scheme of edge servers is completely ignored, and there is no mention of how to use the data obtained from edge computing, although the data is stored persistently. Therefore, we need to propose a better edge computing architecture to solve the scheduling problem of edge servers and to apply the data obtained from edge computing.

2.2. Energy consumption optimization algorithms
After establishing the edge computing architecture, the energy consumption of edge computing needs to be optimized. The problem of edge computing energy consumption optimization has been given by many scholars with their own solutions. Wei et al. [8] constructed a greedy algorithm to select the edge server with the lowest energy consumption for each offloading. Bi et al. [9] divided the energy consumption of edge computing into data upstream transmission energy consumption, data downstream transmission energy consumption and the energy consumption required for computing on the edge servers, and modeled the corresponding energy consumption. Later, PSO (Particle Swarm Optimization) was used to optimize these energy consumptions. Wang et al. [10] used a deep learning approach to deal with the energy consumption optimization problem and they constructed a DNN
(Deep Neural Network) using gradient descent to perform the optimization. Although the greedy algorithm can find the optimal solution in the form of global traversal under small data sets, the efficiency of global access will be very poor along with the increase of the number of edge servers, which is difficult to apply to real business scenarios. Heuristic algorithms such as PSO have a certain degree of improvement in efficiency over the greedy algorithm, but the majority of time is wasted in useless search due to its large search solution space. Moreover, heuristic algorithms usually have no rigorous theoretical proof of convergence, and their stability needs to be considered. As for DNN methods, on the other hand, it has to part data sets as test sets and training sets, on the other hand, it generally used to solve classification and regression problems, and further processing of the output results is required to get decisions for offloading. In conclusion, the currently existing energy consumption optimization schemes for edge computing need to be improved and we need to propose a new solution.

3. Architecture

As shown in Fig. 1, we propose a generic edge computing architecture, which consists of four main components: user layer, scheduler layer, edge computing layer, and data application layer.

3.1. User layer

The user layer mainly consists of various user terminal devices and various types of sensors that are responsible for collecting data and transmitting tasks. For example, wearable devices on the user can be used to collect various physiological data and communicate with the edge computing server by sending heartbeat data packets to achieve user life and health detection. It also includes sensors placed on vehicles for speed, position, and direction to monitor vehicle driving data in real time, thus enabling assisted driving and intelligent navigation. There are also include users' smartphones, tablets, which offload locally tasks such as gaming, video processing, etc. to the edge by installing and deploying computational offloading services. Such devices send the collected information to the edge computing scheduler in real time to feed the edge servers for analysis and computation.

3.2. Scheduler layer

The main task of the scheduler layer is to schedule the edge servers distributed around the users, which is defined as a queue model. In this layer, the data from the computational offloading is encapsulated as a service and added to the request processing queue. After that, each element of the queue leaves the queue one by one and the edge server with lower energy consumption is selected for task processing according to our proposed ECOA.
3.3. Edge computing layer
The edge computing layer consists of a series of edge servers deployed around the users, responsible for handling the computational processing of tasks from the scheduler. On the one hand, since the edge servers are deployed in the vicinity of the user, the energy consumption and computational latency of user data transmission will be reduced. On the other hand, the data avoids being transferred to the cloud server, the user's privacy and security can be guaranteed to a certain extent.

3.4. Data application layer
The main significance of the data application layer is to fully utilize the value of the data. The main task is to present sensitive content from the data processed by the edge computing layer to researchers and analysts. Such as performing data statistics, data analysis, and providing data set for machine learning to train, etc.

4. Model
We assume that there exist $M$ MEC servers, $M = \{1, 2, \ldots, M\}$, and users can offload their tasks to them for processing. We define the set of services as $S = \{1, 2, \ldots, S\}$. These services need to be transmitted with the help of a transport channel during transmission, and we assume that the transport channels can be divided into multiple subcarriers, we use $N = \{1, 2, \ldots, n, \ldots, N\}$ to represent them. We use a triple $T = (D_s, G_s, T_{max})$ to denote the tasks that need to be offloaded. Where $D_s$ is the data size of the task, $G_s$ is the number of CPU clock cycles per bit spent by the task, and $T_{max}$ is the maximum latency requirement that needs to be satisfied for the task to perform computing offloading. Besides, we define the offloading assignment matrix $h_{s,m} \in \{0, 1\}$ to indicate whether the task $s$ is offloaded to the edge server $m$. If $h_{s,m} = 0$ indicates that task $s$ is offloaded to edge server $m$, otherwise task $s$ is not processed on edge server $m$.

4.1. Local computing
In local computing, we define the CPU frequency (i.e., the number of clock cycles per second that the CPU can execute a task) as $f_s$, and when the service $s$ is executed on the local device, its running time can be expressed as follows:

$$T_s = \frac{G_s D_s}{f_s} \quad (1)$$

The energy consumption of local computing can be expressed as the following equation.

$$E_s = k_0 (f_s)^2 G_s \quad (2)$$

where $k_0$ is a coefficient related to the specific architecture and design of the chip. It is typically set as $k_0 = 1.0 \times 10^{-26}$ [11]. Usually, to save energy as much as possible, the CPU clock frequency is adjusted in real time based on voltage and frequency, such as the turbo boost technology of intel and the turbo core technology of AMD.

4.2. Edge computing
When the user's data is offloaded to the edge server for computing, this energy consumption mainly includes the energy consumption during task transmission and the task running on the edge server. Analogously to local computation, we can express the latency spent by the service $s$ to compute in the edge server $m$ by the following equation:
where $f^g_{s,m}$ is the CPU clock frequency of the edge server $m$ executing the service $s$. Correspondingly, we can obtain the energy consumption of the edge server $m$ executing the task $s$ as follows:

$$E^g_{s,m} = k_0 (f^g_{s,m})^2 G_s$$  (4)

According to Shannon's theorem, we can define the data transfer rate from the user's computational offload task $s$ to the edge server $m$ as follows:

$$r_{s,m} = B \sum_{n=0}^{\infty} w_{m,s,n} \log_2 \left(1 + \frac{\alpha_n P_{s,n}}{\delta^2 R_{s,m}}\right)$$  (5)

where $B$ is the channel transmission bandwidth, $P_{s,n}$ is the channel transmission power when sending task $s$, $\alpha_n$ is the channel gain, and $w_{m,s,n} = \{0,1\}$ indicates whether to use this subcarrier for task transmission. When $w_{m,s,n} = 0$ indicates that this subcarrier is used for task transmission, otherwise, this subcarrier is not used. $\delta^2$ is additive Gaussian white noise, which is used to indicate the intensity of channel interference. $R_{s,m}$ is the distance-dependent flat fading component. It can be defined as follows:

$$R_{s,m} = \sqrt{(X_{s,m} - x_s)^2 + (Y_{s,m} - y_s)^2}$$  (6)

where $(x_s, y_s)$ is the location of the UE, and $(X_{s,m}, Y_{s,m})$ is the location where the task $s$ needs to offload to the edge server. Then we can obtain the data transmission delay from task $s$ to edge server $m$:

$$T^{r}_{s,m} = \frac{r_{s,m}}{D_s}$$  (7)

According to the energy consumption equation $E = PT$, we can obtain the energy consumption during the transmission from task $s$ to edge server $m$ as follows:

$$E^{r}_{s,m} = T^{r}_{s,m} P_{s,n}$$  (8)

The total delay in the task $s$ computational offloading is the sum of the transmission delay and the calculation delay, which we express using the following equation:

$$T_{s,m} = T^g_{s,m} + T^{r}_{s,m}$$  (9)

So the energy consumption of the next task $s$ for computational offloading as a whole can be expressed as:

$$E_{s,m} = E^g_{s,m} + E^{r}_{s,m}$$  (10)

Ultimately, the problem of optimizing energy consumption and resource allocation for the overall computational offload can be modeled as follows:
\[ \min \sum_{i=1}^{S} \sum_{m=1}^{M} h_{i,m} E_{i,m} \]

S1.

C1: \( h_{i,m} = \{0,1\} \)

C2: \( \sum_{m=1}^{M} h_{i,m} = 1 \)

C3: \( w_{m,s,n} = \{0,1\} \) (11)

C4: \( \sum_{s=1}^{S} \sum_{m=1}^{M} w_{s,m,n} = 1 \)

C5: \( 0 < f_{s,m} < F_{\text{max}} \)

C6: \( 0 < w_{m,s,n} P_{s,m} < P_{\text{max}} \)

C7: \( \sum_{m=1}^{M} h_{i,m} (T_{r,m}^{g} + T_{r,m}^{p}) < T_{\text{max}} \)

where C1 indicates whether the task \( s \) is offloaded to the edge server \( m \). C2 ensures that each edge computing server can only accept no more than one task at a time. C3 indicates whether the service \( s \) is transmitted to the edge server \( m \) over the subcarrier \( n \). C4 ensures that each subcarrier is assigned exclusively to one user. C5 specifies that the range of CPU clock frequencies in the edge server should be an integer and less than a given value. Similarly, C6 specifies that the range of channel transmission gain should be an integer and less than a given value. And then C7 ensures the time of each offloaded task no more than deadline. It is easy to notice that the problem is mixed integer nonlinear programming (MINLP), which is characterized by an exponential increase in the complexity of the algorithm as the number of UEs increases. Traditional methods generally decompose it into several subproblems and then deal with each subproblems, which is very complex and inefficient. Next, we explore the reinforcement learning-based solution.

5. Method

5.1. Markov decision process

We usually use the Markov decision process to describe the learning process of an agent in reinforcement learning. It is usually represented as a five-tuple: \( E = \langle S, A, R, P, \gamma \rangle \). Where \( E \) is the environment where the agent learns in, \( A \) is the action that the agent makes to the environment, and \( R \) is the gain that the agent obtains from the environment after taking an action on it. \( P \) is the probability that the agent makes a state transfer, and \( \gamma \) is the decay rate of the cumulative gain obtained by the agent. Specifically, it denotes that the agent observes the state \( s_{t} \in S \) from the environment through continuous interaction with the complex environment and makes an action \( a_{t} = \pi(a_{t}, s_{t}) \), \( a_{t} \in A \) according to a certain strategy. Then the environment gives the agent a reward \( r_{t} = r(a_{t}, s_{t}) \), \( r_{t} \in R \). The agent will acquire a new state from the environment based on the state transfer probability. The system will cycle through this series of operations. We use \( U(t) \) to record the cumulative gain of the agent, which usually needs to be multiplied by the decay rate to weaken the future gain as follows:

\[ G_{t} = R_{r+1} + \gamma R_{r+2} + \gamma^{2} R_{r+3} + \ldots = R_{r+1} + \gamma (R_{r+2} + \gamma R_{r+3} + \ldots) \] (12)

And then we can define the state action value function:
In this paper, we assume that the agent is decision scheduler for offloading edge computing. Depending on the specifics of edge computing, Action, State and Reward are defined as follows:

Actions: The service needs to offload subcarrier \( n \) to the edge server \( m \), which are denoted like this:

\[
A = \{a_0, a_1, a_2, \ldots, a_N\}
\]

(14)

States: We define the environment in which the agent located as follows:

\[
S = \{s_1, s_2, \ldots, s_M\}
\]

(15)

where \( s_m \) denotes the computational decision.

Reward: In general, the reward that an agent obtains from the environment is related to the objective function, so we define the reward of an agent taking action on the environment as

\[
R(a_t, s_t) = -E(a_t, s_t)
\]

where \( E(a_t, s_t) \) is used to evaluate the energy consumption in the current state.

5.2. Q-learning based solution

Q-learning is a typical reinforcement learning method that automatically learns in an edge-computing system according to the MDP parameters that have been set. An action-state pair is obtained in each step according to the guide of the reward function. The value \( Q(s_t, a_t) \) is usually used to represent an action-state pair, and these values form a Q-table. The algorithm continuously updates the data in the Q-table until the Q values obtained in each state converge to the optimal one. By executing the Q-function, the cumulative gain can be obtained as follows: the update method of the Q-function is generally expressed as follows [13]:

\[
Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max Q(s_{t+1}, a_{t+1})]
\]

(16)

where \( \alpha \) is the learning rate and \( 0 < \alpha < 1 \), and \( \gamma \) is the decay rate, indicating the degree of acceptance of future rewards and \( 0 < \gamma < 1 \). For the selection strategy of action, we usually have two strategies: exploration and exploitation. Exploration refers to the random selection of actions under the state \( s_t \), and exploitation refers to the full use of existing Q values to select the action corresponding to the optimal Q value. We usually use \( \epsilon \) to denote the probability of exploration, and to make it dynamically adaptable to different stages of the cycle, we define it as a variable.

\[
\epsilon = e^{\frac{-t}{T}}
\]

(17)

where \( k \) is the number of current cycles and \( T \) is the total number of cycles. When the probability is lower than \( \epsilon \), the agent will choose the action through the strategy of exploration, and when the probability is higher than \( \epsilon \), the agent will choose the action through the strategy of exploitation. We can see that when the agent is at the beginning of the cycle, the agent is more inclined to learn new knowledge. As the number of cycles rises, the opportunity to exploit gradually increases, which will facilitate the algorithm to achieve convergence.

We propose ECOA based on the Q-Learning algorithm. It works like this: First, initialize the Q-table. And then execute the cycle according to the count of training. For each cycle, first determine \( \epsilon \) according to (17) to choose the corresponding policy, select the action and execute it. The environment will feedback the state based on the reward function. Finally, the Q-table is updated based on these states according to (16). The specific execution process is shown in Algorithm 1.
6. Simulation and performance analysis

6.1. Parameter setting
In this section, we start the simulation of the proposed algorithm, where the parameters are given in Table 1, in which radius $R_j$ for all edge computing servers is set to $r = 600m$, the channel bandwidth is set to $B = 12.5kHz$, the noise variance is set to $\delta^2 = 10^{-13}W$, the input-data size is select from the interval of $[1000,1200]bits$, the computation workload is set to $G_t = 1000cycles/bit$, the maximum accomplished deadline is set to $T_{max} = 9ms$, the CPU frequency of UEs is set to $f_{u}^{c} = 0.6GHz$, the CPU frequency of edge computing servers is select from the interval of $[1.1,1.2]GHz$. We compare the ECOA against the following offloading algorithm:

1) Minimum Distance Offloading Algorithm (MDOA). It means that the UEs offload their tasks to the nearest edge computing server.

2) Random Offloading Algorithm (ROA). Each UE randomly selects the edge computing server to offload services.

6.2. Energy consumption with the increase of services
In Figure 3, we compare the change in energy consumption of each algorithm with the number of UEs increases. We can see that the total energy consumption of edge computing tends to increase with the increase of the number of UEs, but our proposed ECOA has the lowest energy consumption. ECOA can dynamically make computational offloading decisions according to environmental conditions. In addition, the dynamically changing greediness also makes the algorithm adapt to the environment to a great extent. So the model tends to choose the edge server with lower energy consumption for computational offloading.
6.3. Energy consumption with the increase of edge computing servers

In Figure 4, we compare the change in energy consumption of each algorithm as the number of edge computing servers increases for a constant number of services. We can see that compared to other algorithms, the overall energy consumption of ECOA remains stable along with the increase in the number of edge servers, which proves the effectiveness of the algorithm.

7. Conclusion

In this paper, we propose a four-layer edge computing architecture and propose the ECOA to optimize the energy consumption problem in edge computing. Finally, we compare ECOA with three benchmark algorithms to prove the effectiveness of the algorithm. In the future, we will explore more solutions for optimize the energy consumption of edge computing and try to apply and deploy ECOA in more realistic environments.

| Parameters                          | Settings |
|-------------------------------------|----------|
| Radis $R_j$ for all edge computing servers | $600m$   |
| Bandwidth                           | $12.5khz$|
| Noise variance $\delta^2$           | $10^{-13}w$|
Input-data size $D_s$ & 1000–1200 bits

The computation workload/intensity $G_s$ & 1000 cycles/bit

Maximum accomplished deadline $T_{max}$ & 9 ms

The CPU frequency of UEs $f^u_{x,m}$ & 0.6 GHz

The CPU frequency of edge computing servers $f^e_{x,m}$ & 1.1–1.2 GHz

References

[1] infoobs, "The number of global IOT devices is growing rapidly," 2021.https://www.infoobs.com/article/20210119/44884.html, Last accessed on 2021-12-13.

[2] M. Yao, L. Chen, T. Liu and J. Wu, "Energy Efficient Cooperative Edge Computing with Multi-Source Multi-Relay Devices," *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2019, pp. 865-870, doi: 10.1109/HPCC/SmartCity/DSS.2019.00126.

[3] J. M. Khurpade, D. Rao and P. D. Sanghavi, "A Survey on IOT and 5G Network," *2018 International Conference on Smart City and Emerging Technology (ICSCET)*, 2018, pp. 1-3, doi: 10.1109/ICSCET.2018.8537340.

[4] H. Lv, D. Chen and Y. Wang, "Deployment of Edge-Computing in 5G NFV Environment and Future Service-Based Architecture," *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, 2018, pp. 811-816, doi: 10.1109/CompComm.2018.8780937.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[6] L. Li, Z. Kuang and A. Liu, "Energy Efficient and Low Delay Partial Offloading Scheduling and Power Allocation for MEC," *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1-6, doi: 10.1109/ICC.2019.8761160.

[7] N. Abbas, Y. Zhang, A. Taherkordi and T. Skeie, "Mobile Edge Computing: A Survey," in *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450-465, Feb. 2018, doi: 10.1109/JIOT.2017.2750180.

[8] F. Wei, S. Chen and W. Zou, "A greedy algorithm for task offloading in mobile edge computing system," in *China Communications*, vol. 15, no. 11, pp. 149-157, Nov. 2018, doi: 10.1109/CC.2018.8543056.

[9] J. Bi, H. Yuan, S. Duanmu, M. Zhou and A. Abusorrah, "Energy-Optimized Partial Computation Offloading in Mobile-Edge Computing With Genetic Simulated-Annealing-Based Particle Swarm Optimization," in IEEE Internet of Things Journal, vol. 8, no. 5, pp. 3774-3785, 1 March1, 2021, doi: 10.1109/JIOT.2020.3024223.

[10] L. Wang, X. Sun, R. Jiang, W. Jiang, Z. Zhong and D. W. Kwan Ng, "Optimal Energy Efficiency for Multi-MEC and Blockchain Empowered IoT: a Deep Learning Approach," *ICC 2021 - IEEE International Conference on Communications*, 2021, pp. 1-6, doi: 10.1109/ICC42927.2021.9500558.

[11] J. Zhang et al*, "Energy-Latency Tradeoff for Energy-Aware Offloading in Mobile Edge Computing Networks," in *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633-2645, Aug. 2018, doi: 10.1109/JIOT.2017.2786343.

[12] B. Liu, C. Liu and M. Peng, "Resource Allocation for Energy-Efficient MEC in NOMA-Enabled Massive IoT Networks," in *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 4, pp. 1015-1027, April 2021, doi: 10.1109/JSAC.2020.3018809.
[13] K. Jiang, H. Zhou, D. Li, X. Liu and S. Xu, "A Q-learning based Method for Energy-Efficient Computation Offloading in Mobile Edge Computing," 2020 29th International Conference on Computer Communications and Networks (ICCCN), 2020, pp. 1-7, doi: 10.1109/ICCCN49398.2020.9209738