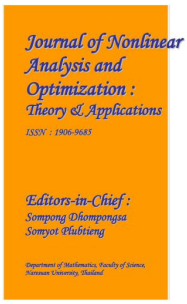
Journal of **N**onlinear **A**nalysis and **O**ptimization
Vol. 15, Issue. 1, No.15 : 2024
ISSN : **1906-9685**



Paper ID: ICRTEM24_127 **ICRTEM-2024 Conference Paper**

ENERGY USAGE ENHANCING EDGE COMPUTING THROUGH REINFORCEMENT LEARNING

[#1]**BURLA SRINIVAS,** ***Research Scholar,***
[#2]**Dr. PAWAN KUMAR,** ***Associate Professor & Guide,***
***Department of Computer Scinece &Engineering*,**
NIILM UNIVERSITY, KAITHAL, HARYANA, INDIA.

ABSTRACT: When users are remote from cloud servers, a phenomena known as over-centralization occurs in cloud computing. Long delays and high computational energy consumption are related with user-cloud communication. Edge computing is currently a prominent topic in academic research. By pushing cloud computing nodes to the edge, users can assign jobs directly to edge servers. Cloud computing isn't as user-friendly as edge computing. When consumers and edge servers communicate, they consume less energy and have a shorter transmission latency. This study will primarily look at the core architecture of edge computing in order to better leverage its benefits and support its growth. Furthermore, we describe a reinforcement learning-based method for optimizing the energy usage of edge computing. Finally, we use simulated trials to compare the effectiveness of our suggested technique to other schemes.

Keywords: Edge Computing,Energy Consumption,Optimization,Edge Devices

1.INTRODUCTION

The number of IoT devices on the planet is rapidly expanding, with more than 24.6 billion linked devices expected to exist by 2025. This figure is extremely remarkable. The rapid proliferation of these devices makes it difficult to estimate data processing capacities. Previously, we would upload work, data, and other materials to cloud servers and wait for the computers to process our requests while using cloud computing solutions. We encountered lengthy wait times, unbearable connection instability, and numerous security and privacy breaches. Placing servers closer to the edge, which means closer to the user, is currently a hot topic in academics. We commonly refer to it as edge computing. It enables users to move the processing of complicated services from local devices to a nearby edge server. The decreased transmission distance significantly reduces the communication delay between the user and the server. Thus, it can meet the low latency requirements of IoT devices. Edge computing improves user data security by storing data close to the user and preventing transmission to centralized servers.

1.2. High energy consumption of 5G

5G mobile networks will improve data transfer by offering faster speeds, more bandwidth, and lower latency.5G services rely on servers designed with

a service-oriented architecture. Different types of edge computing are likely to be directly integrated in the future.

In the design of the 5G network. This will expand the possibilities for edge computing. However, reality may not always reflect our idealized ideas. The high frequency of 5G data transmission consumes a substantial amount of energy. Furthermore, the edge server performs the user's duties, which requires additional energy for calculation and data processing. Edge computing will only be possible in specialized circumstances because to its high cost and significant energy requirements, posing obstacles for wider adoption across multiple sectors.

In 2018, global CO2 emissions were reported to be 34,041,045,974 tons. To save money and energy, we must devise a practical strategy to improving the energy efficiency of edge computing.

1.3. Contributions

Our primary contributions to this work are as follows: (1) We develop a four-tier edge computing architecture to enable user computation offloading. To reduce energy consumption in edge computing, we offer an Energy Consumption Optimization approach (ECOA) based on the Q-learning method. The method's usefulness is validated by comparing it to other algorithms. The ensuing sections of the document are organized in the following way: In section 2, we provide a brief overview of the relevant studies. Part 3 focuses on the edge computing architecture. The energy consumption model is described in Section 4. Section 5 shows illustrations of the Markov decision process and Q-learning-based ECOA. The method's efficiency is then demonstrated via simulated studies using six instances. Section 7 provides a conclusion and overview of the work.

2.REALTED WORK

2.1. Edge computing architectures

Several scientists have created edge computing architectures with tiered topologies that enable the isolation and disconnection of specific levels. Li et al. present a simple two-tier edge computing architecture. The User Equipment (UE) is the lowest layer that collects data and comprises a wide range of mobile devices. The layer above is known as edge computing, and it consists of a large number of edge servers that manage compute and job execution. The UEs delegate computationally demanding tasks to the edge servers. Furthermore, Abbas et al. provide an alternate edge computing architecture design scenario. The two-tier architecture is enhanced to include a cloud data center layer that continually saves data at the edge computing level. Current edge computing devices can handle some but not all edge computing activities. The advanced scheduling plan for edge servers is generally ignored, and little guidance is given on how to best use the data protected by edge computing. To address the scheduling issue with edge servers and make use of the data collected by edge computing, an enhanced edge computing architecture must be developed..

2.2. Energy consumption optimization algorithms

Developing the architecture for edge computing is crucial to optimizing the technology's energy consumption. Many scholars have disputed the optimization of energy usage in edge computing and proposed alternative solutions. Wei et al. created a greedy algorithm to select the edge server with the lowest energy consumption for each offloading. Bi et al. classified edge computing energy usage into three categories: energy required for processing on edge servers, energy spent transferring data upstream, and energy spent on transmission downstream. PSO was used to optimize energy consumption. Wang et al. employed deep learning to address the difficulty of optimizing energy consumption. They created a DNN (Deep Neural Network) and employed gradient descent for optimization. The greedy method can identify the global traversal optimal solution for small data sets, but as the number of edge servers increases, global access efficiency rapidly declines, making it impossible to employ in practical commercial applications. Although heuristic algorithms like PSO are

slightly more efficient than greedy algorithms, their huge search solution space means that the majority of the time is wasted searching. It is also important to assess the stability of heuristic algorithms, as they frequently lack robust theoretical demonstrations of convergence. However, DNN approaches necessitate the partitioning of data into test and training sets. They are widely employed to tackle classification and regression problems, and additional processing of the output results is required before drawing conclusions about offloading. To summarize, we must improve existing energy consumption optimization methods for edge computing and present a novel technique.

3. ARCHITECTURE

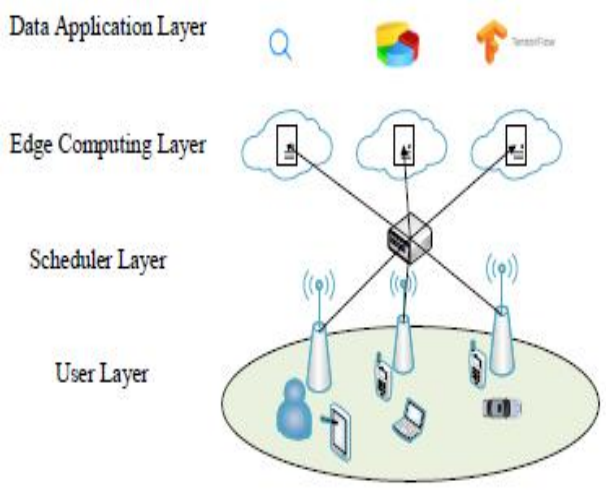


Figure 1 Edge Computing Architecture

As shown in Figure 1, we propose a comprehensive edge computing framework made up of four major components: the user layer, the scheduler layer, the edge computing layer, and the data application layer.

User layer

The user layer comprises of different user terminals and sensors that collect data and provide commands. Wearable devices worn by the user can gather various physiological data and send packets of heartbeat data to an edge computing server for analysis of the user's well-being and health state. Sensors are mounted on vehicles to track position, speed, and direction in real time, allowing for aided driving and intelligent navigation. Furthermore, some consumers use computational offloading services on their smartphones and tablets to move locally-based tasks like gaming and video processing to the edge. The devices deliver the acquired data in real time to the edge computing scheduler, which uses it for analysis and processing on the edge servers.

Scheduler layer

The primary function of the scheduler layer, also known as a queue model, is to arrange the scheduling of edge servers spread among customers. This layer holds the service-encapsulated data from the computational offloading for request processing. Following our proposed ECOA, the edge server with the lowest energy consumption is chosen to process tasks as they are removed from the queue individually.

Edge computing layer

The edge computing layer is made up of edge servers that are located near users and perform the computational processing of jobs from the scheduling system. The proximity of edge servers to users will reduce energy consumption and computational delay in user data transmission. However, because the data remains on the local server and is not sent to the cloud, the user's security and privacy are somewhat protected.

Data application layer

The fundamental purpose of the data application layer is to maximize the value of the data. The primary task is to give researchers and analysts access to sensitive content derived from data handled by the edge computing layer. Examples include performing data analysis and statistics, as well as providing training data sets for machine learning, among other jobs.

4. MODEL

Users can assign jobs to M MEC servers (M = {1,2,..M}) for processing. S = {1, 2,..., s,..., S} represents the defined services. Transmitting these services requires the use of a transport channel. Consider the transport channel as a set of subcarriers, indicated by N = {1, 2,..., n, N}. We

define the jobs for offloading using a triple s max, where max T indicates the maximum delay required for compute offloading to occur, s D is the task's data size, and s G is the number of CPU clock cycles per bit utilized by the work. We utilize the {0,1} s m h = offloading assignment matrix to determine if work s is assigned to edge server m. Task s is passed to edge server m only if it is less than or equal to m; else, task s is not processed by edge server m.

Local computing

In local computing, the CPU frequency is denoted as l s f, which represents the number of clock cycles per second that the CPU can do a task. When executed, the operation time of service(s) on the local device can be characterized as:

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

The above equation may represent the amount of energy consumed by local computing.

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

The coefficient associated with the chip's specific architecture and design is denoted as 0k. The value is usually set to 26 x 10^0 k = 1.0*(10^-).To improve energy efficiency, the CPU clock frequency is typically dynamically modified in response to voltage and frequency. Examples of this include AMD's Turbo Core technology and Intel's Turbo Boost technology.

Edge computing

Energy consumption during task transmission and execution on the edge server is mostly caused by offloading user data for computation. The equation below can be used to characterize the latency experienced by service s when computing on edge server m, which is analogous to local computation.

$$T_{s,m}^g = \frac{G_s D_s}{f_{s,m}^g} \quad (3)$$

What is the CPU clock frequency of edge server m while it is running service ,As a result, we can calculate the energy consumption of edge server m as it performs job s in the following way:

$$E_{s,m}^g = k_0 (f_{s,m}^g)^2 G_s \quad (4)$$

The data transfer rate from the user's computational offload task s to the edge server m can be calculated using Shannon's theorem.

$$r_{s,m} = B \sum_{n=1}^{N} w_{m,s,n} log_2 (1 + \frac{\alpha_n P_{s,n}}{\delta^2 * R_{s,m}}) \quad (5)$$

The variables wm, s, and n are binary indications used to pick a subcarrier for task transmission. B is the channel bandwidth, Ps,n is the power for task s, and αn is the channel gain. This subcarrier is only used for task transmission when wm, s, and n are equal to zero; otherwise, it is not used. The degree of channel interference can be quantified using additive Gaussian white noise (Θ 2). R s,m represents the flat fading component, which changes with distance. The definition goes as follows:

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

The UE's position is defined by (xs, ys), while the location where the work must be offloaded to the edge server is denoted by (X sm, Y sm). Thus, the data transmission latency between task s and edge server m can be computed.

$$T_{s,m}^p = \frac{r_{s,m}}{D_s} \quad (7)$$

To compute the energy consumption during the transmission from task s to edge server m, we can apply the equation E = PT

$$E_{s,m}^p = T_{s,m}^p P_{s,n} \quad (8)$$

The total delay in computational offloading for a task can be estimated by summing the transmission and calculation delays and applying the provided equation.

$$T_{s,m} = T_{s,m}^g + T_{s,m}^p \quad (9)$$

The entire energy consumption necessary for the upcoming computational offloading tasks is summarized as follows:

$$E_{s,m} = E_{s,m}^g + E_{s,m}^p \quad (10)$$

Finally, the challenge of optimizing resource allocation and energy usage for the whole computing workload can be represented as:

$$\begin{aligned}
&min\sum_{i=1}^{S}\sum_{m=1}^{M}h_{s,m}E_{s,m}\\
&s.t.\\
&C1: h_{s,m}=\{0,1\}\\
&C2: \sum_{m=1}^{M}h_{s,m}=1\\
&C3: w_{m,s,n}=\{0,1\}\\
&C4: \sum_{s=1}^{S}\sum_{m=1}^{M}w_{s,m,n}=1\\
&C5: 0<f_{s,m}^{g}<F_{max}\\
&C6: 0<w_{m,s,n}P_{s,n}<P_{max}\\
&C7: \sum_{m=1}^{M}h_{s,m}(T_{s,m}^{g}+T_{s,m}^{p})<T_{max}
\end{aligned} \quad (11)$$

C1 indicates whether the edge server is used to offload the operation. C2 assures that each edge computing server can only handle one task at a time. C3 denotes the services transmitted via subcarrier n to edge server m.C4 ensures that each subcarrier is assigned to an individual user. C5 states that the edge server's CPU clock frequency range must be an integer less than a predetermined value. Similarly, C6 states that the channel transmission gain range must be an integer less than a certain threshold. C7 ensures that all offloaded assignments are completed before the deadline. The problem is classified as mixed integer nonlinear programming (MINLP), which is distinguished by a large rise in algorithmic complexity as the number of UE (User Equipment) grows. Traditional techniques typically divide the problem into smaller concerns and address each one separately, which is extremely complex and wasteful. The solution will be analyzed using reinforcement learning.

5. METHOD

Reinforcement learning frequently uses the Markov decision process to describe how an agent learns. Usually shown as a five-tuple. E =< S, A, R, P,γ > . A represents the agent's alteration of the environment, R represents the reward the agent receives from the environment as a result of the modification, and E represents the agent's learning environment. P represents the likelihood of the agent switching between states. γ is the decay rate of the cumulative gain obtained by the agent. Specifically, it denotes that the agent observes the state ts ∈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 obtain a new state from its surroundings based on the state transfer probability. The system will repeat this series of steps. We track the agent's total gain using U(t), which is frequently adjusted by the decay rate to reduce future gain.

$$\begin{aligned}
G_t&=R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\ldots=R_{t+1}+\gamma(R_{t+2}+\gamma R_{t+3}+\ldots)\\
&=R_{t+1}+\gamma G_t
\end{aligned} \quad (12)$$

The subsequent step is to define the state action value function, as follows:

$$\begin{aligned}
q_\pi(s,a)&=\mathbb{E}_\pi[G_t\,|\,A_t=a,S_t=s]\\
&=\mathbb{E}_\pi[R_{t+1}+\gamma q_\pi(S_{t+1},A_{t+1})\,|\,A_t=a,S_a=s]
\end{aligned} \quad (13)$$

This paper refers to the agent as an edge computing outsourced decision scheduler. Action, State, and Reward are defined as follows, depending on the characteristics of edge computing:

Actions: The service unloads subcarriers to the peripheral servers, which are defined as follows:

$$A=\{a_{00},a_{01}\ldots a_{0N},a_{1N}\ldots a_{MN}\} \quad (14)$$

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

$$S_t=\{s_1,s_2\ldots s_M\} \quad (15)$$

where m s denotes the computational decision.

Reward: We define an agent's reward when acting on the environment as follows because, in general, the reward received from the environment is related to the objective function. $R = -E(a_t, t_t)$. where $E(a_t, s_t)$ is used to evaluate the energy consumption in the current state.

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 theMQ-table until the Q values obtained in each state converge to the optimal one. By executing the Qfunction, the cumulative gain can be obtained as follows: the update method of the Q-function is generally expressed as follows

$$Q(s_t,a_t) = (1-\alpha)Q(s_t,a_t) + \alpha[R_{t+1} + \gamma max Q(s_{t+1},a_t)] \quad (16)$$

where α is the learning rate and $0 < \alpha < 1$, and γ 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 t_s, and exploitation refers to the full use of existing Q values to select the action corresponding to the optimal Q value. We usually use to denote the probability of exploration, and to make it dynamically adaptable to different stages of the cycle, we define it as a variable.

$$\epsilon = e^{\frac{e^{-T}-k}{1-e^{-T}}} \quad (17)$$

T denotes the total number of cycles, while k represents the number of cycles that are currently in use. The agent will use the exploration strategy when the likelihood is less than or equal to Ɛ, and the exploitation technique when the probability is greater than or equal to Ɛ. The agent is more likely to learn new information at the start of the cycle. As the number of cycles increases, the algorithm will achieve convergence more quickly due to increased opportunities to exploit. We propose ECOA using the Q-Learning method. Here's how it works: Start by initializing the Q table. Execute the cycle using the set number of repetitions from the training. Use equation (17) to calculate Ɛ for each cycle before selecting and implementing the appropriate policy and action. The reward function determines how the environment responds to the state. Finally, following equation (16), the Q-table is updated to reflect these states. Algorithm 1 shows the exact execution procedure.

Algorithm 1: Energy Consumption Optimization Algorithm

Input: cycle-index T, state set S, action Set A, learning rate α, discount factor γ.

Output: Q-table

Initialize $Q(s_t, a_t) = \{0\}$;
for $k = 0$; $k < T$; k+=1 do
  Initialize s_t;
  Randomly generated variable p.
  if $p < e^{\frac{e^{-T}-k}{1-e^{-T}}}$ then
    Random select a_t from A;
  end
  else
    Select the highest Q value as a_t;
  end
  Execute action a_t, observe the reward r_t and the next state s_{t+1};
  Update $Q(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha[R_{t+1} + \gamma max Q(s_{t+1}, a_t)]$;
  $S_t = S_{t+1}$;
end

Figure 2. Energy Consumption Optimization Algorithm

6. SIMULATION AND PERFORMANCE ANALYSIS

Parameter setting

In this section, we start the simulation of the proposed algorithm, where the parameters are given in Table 1, in which radis Rj 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 s = 1000 cycles bit , the maximum accomplished deadline is set to

$T_{max} = 9ms$ the CPU frequency of UEs is set to $f_{s,m}^{g} = 0.6GHz$ 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). Rach UEs randomly select the edge computing server to offload services.

Energy consumption with the increase of services

Figure 3 depicts the change in energy consumption for each technique as the number of UEs grows. Edge computing consumes more energy as the number of devices increases. However, our proposed ECOA has the lowest energy consumption. ECOA can dynamically select computational offloading based on external conditions. Furthermore, the algorithm's high level of environmental adaptability is aided by a dynamically shifting amount of greediness. As a result, in computational offloading, the model frequently selects the edge server with the lowest energy consumption.

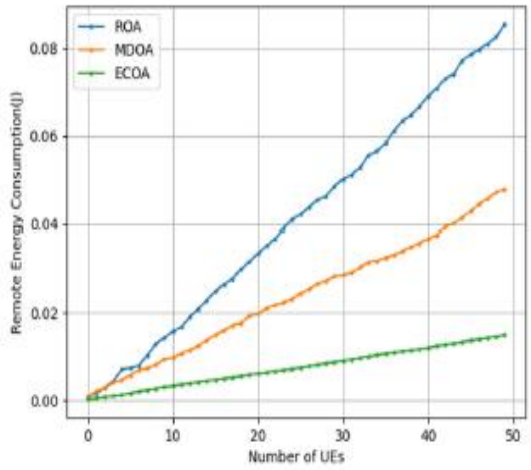


Figure 3. Energy consumption versus the number of UEs

Energy consumption with the increase of edge computing servers

Figure 4 depicts the change in energy usage for each algorithm as the number of edge computing servers increases, for a given number of services. ECOA's overall energy consumption remains constant as the number of edge servers grows, setting it apart from other algorithms and demonstrating its effectiveness.

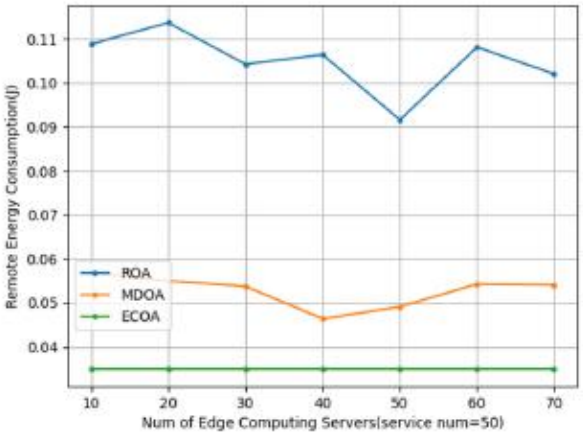


Figure 4. Energy comsumption versus the number of edge computing servers

7. CONCLUSION

This work introduces a four-layer architecture and the ECOA for addressing energy consumption concerns in edge computing. against demonstrate ECOA's effectiveness, we shall compare it against three benchmark algorithms. In the future, we will look at new ways to reduce energy consumption in edge computing and try to implement ECOA in more realistic scenarios.

Table 1. Simulation settings

Parameters	Settings
Radis R_j for all edge computing servers	$600m$
Bandwidth	$12.5khz$
Noise variance δ^2	$10^{-13}w$

Input-data size D_s	$1000-1200bits$
The computation workload/intensity G_s	$1000cycles/bit$
Maximum accomplished deadline T_{max}	$9ms$
The CPU frequency of UEs $f_{s,m}^{l}$	0.6GHz
The CPU frequency of edge computing servers $f_{s,m}^{s}$	$1.1-1.2GHz$

REFERENCES

1. infoobs, "The number of global iot devices is growing rapidly,"
2. M. Yao, L. Chen, T. Liu and J. Wu, "Energy Efficient Cooperative Edge Computing with Multi- Source Multi-Relay Devices,"
3. J. M. Khurpade, D. Rao and P. D. Sanghavi, "A Survey on IOT and 5G Network,"
4. H. Lv, D. Chen and Y. Wang, "Deployment of Edge-Computing in 5G NFV Environment and Future Service-Based Architecture,"
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
7. and Power Allocation for MEC,"
8. N. Abbas, Y. Zhang, A. Taherkordi and T. Skeie, "Mobile Edge Computing: A Survey,"
9. F. Wei, S. Chen and W. Zou, "A greedy algorithm for task offloading in mobile edge computing system,"
10. 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
11. 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 LearningApproach

.