Hindawi
Security and Communication Networks
Volume 2021, Article ID 5109163, 16 pages
https://doi.org/10.1155/2021/5109163

Research Article

Edge Server Placement for Service Offloading in Internet of Things

Rong Ma

Basic Teaching Department, Nanjing University Jinling College, Nanjing, 210089, China

Correspondence should be addressed to Rong Ma; 030239@jlxy.nju.edu.cn

Received 6 August 2021; Revised 6 September 2021; Accepted 13 September 2021; Published 30 September 2021

Academic Editor: Xuyun Zhang

With the rapid development of the Internet of Things, a large number of smart devices are being connected to the Internet while the data generated by these devices have put unprecedented pressure on existing network bandwidth and service operations. Edge computing, as a new paradigm, places servers at the edge of the network, effectively relieving bandwidth pressure and reducing delay caused by long-distance transmission. However, considering the high cost of deploying edge servers, as well as the waste of resources caused by the placement of idle servers or the degradation of service quality caused by resource conflicts, the placement strategy of edge servers has become a research hot spot. To solve this problem, an edge server placement method orienting service offloading in IoT called EPMOSO is proposed. In this method, Genetic Algorithm and Particle Swarm Optimization are combined to obtain a set of edge server placements strategies, and Simple Additive Weighting Method is utilized to determine the most balanced edge server placement, which is measured by minimum delay and energy consumption while achieving the load balance of edge servers. Multiple experiments are carried out, and results show that EPMOSO fulfills the multiobjective optimization with an acceptable convergence speed.

1. Introduction

Internet of Things (IoT) is a network that connects any object to the Internet through sensors to realize intelligent identification, tracking, and control. With the rapid development of information technology, the IoT is playing an increasingly important role in daily life [1–3]. In recent years, with the popularity of smart mobile devices and the explosion of computationally intensive mobile applications, the lack of computing resources of smart mobile devices and sensors cannot guarantee the real-time processing of these computationally intensive tasks [4, 5]. As a strategy to alleviate the pressure on computing resources, cloud computing is introduced into the IoT [6–8]. The data collected by sensors or computing-intensive tasks from smart devices are transmitted to a cloud platform with powerful storage and computing capabilities, where the computation results can be stored in the cloud for subsequent operations [9, 10].

However, considering long distance between sensors and cloud platform, the transmission delay is unacceptable in some services, like the real-time identify, track, and control [5, 11]. To reduce the transmission delay and improve the quality of service and user experience, edge computing is introduced to reduce the delay and realize real-time control with edge servers, which are closer to user devices. In detail, edge service technology assigns computing and storage capabilities to edge servers and provides edge services with lower latency and better user experience [12, 13].

In most of the existing edge computing research, researchers are more inclined to focus on the process of migrating tasks to the edge server under the premise that the edge server has been deployed. In this process, service providers tend to deploy small number of edge servers for the high price of edge servers and environmental reasons [14, 15]. However, fewer studies are focusing on the layout of edge servers on the effect of edge services while the location of edge servers has a crucial impact on latency [15]. Inefficient edge server layout will cause unacceptable latency and make poor quality of user experience. Therefore, it is necessary to design an efficient edge server layout strategy to ensure the quality of edge services [16]. Not only that, edge servers are not always deployed around the sensors, and it is also necessary to ensure that the data from relatively far away sensors can be processed in time. In addition, to ensure the quality of edge services and the stability of the edge server system and avoid excessive load on some edge servers while other edge servers are not fully utilized, it is urgent to achieve overall load balancing of edge servers. Considering the above requirements, it is a challenge to find an edge server layout strategy that can realize real-time control and guarantee the overall edge service quality.

To solve above challenges, an edge server placement strategy is designed , which aims to reduce the delay in the task transmission and the energy consumption of the edge servers. Specially, the main contributions can be concluded as follows:

(i) We propose an edge server placement method named EPMOSO in which Genetic Algorithm and Particle Swarm Optimization are effectively combined to obtain a set of placement strategies, and most balanced edge server placement considering delay and energy consumption is determined by Simple Additive Weighting Method.

(ii) Several experiments have been carried out, and results prove that the method achieves multiobjective optimization with acceptable convergence speed.

The rest of the article is organized as follows: Section 2 describes the related work. In Section 3 and Section 4, an edge computing system model for offloading services under the IoT environment and the layout strategy of edge servers are described. Massive experiments are conducted in Section 5. In Section 6, we conclude this article.

2. Related Work

In this section, the related work is divided into two parts: (1) disadvantages of cloud computing remote services solved by edge server placement, and (2) research status of edge service placement and service offloading:

(1) A large amount of data generated by sensors and devices in the IoT needs to be processed and stored. However, the central data center cannot meet such high demand for processing and storage resources [17, 18], introducing cloud computing into the IoT, which brings new changes to the data processing and storage. Hong et al. [19] studied whether the cloud system in the IoT can provide a unified platform for the continuous processing of complex data, analyzed the requirements for the engineering design of the IoT cloud system, and discussed the main engineering principles that need to be implemented. Dinh et al. [20] proposed an interactive model that integrates location-based IoT and cloud services to process cloud computing applications. In this model, the IoT cloud system provides sensing services based on mobile users' interests and locations. In addition, in response to the problem of privacy leakage in data transmission in the IoT, Christos et al. [21] integrated the IoT cloud platform and big data and built a security wall between the cloud server and the Internet to eliminate the problem of privacy leakage. However, when dealing with delay-sensitive tasks, the long delay caused by the long physical distance between the sensor and the cloud platform is unacceptable, which promotes the generation and development of edge computing to meet the needs of these delay-sensitive tasks [22]. Edge computing deploys Edge Servers (ES) with rich computing resources and storage resources on the edge of the network to process these delay-sensitive tasks, which effectively reduces delays and the pressure of transmission, and relieve the resource pressure of sensors and equipment [23, 24]. It has become a new trend to integrate edge computing and cloud computing into the IoT to deal with many computing tasks. Hassan et al. [25] put forward the key requirements for the application of edge computing in the IoT and discussed the scenarios where edge computing can be applied. To improve the security and efficiency of the IoT cloud platform, Wang et al. [8] integrated the trust evaluation mechanism, service template, and edge computing into the IoT cloud platform framework. The framework establishes a service parameter template on the cloud platform and establishes a service parsing template in the edge platform to improve the efficiency of the framework and also improves the security of the entire system through the trust evaluation mechanism. In addition, in order to improve the flexibility of the edge physical network platform, Morabito et al. [26] studied how to introduce Lightweight Virtualization (LV) into the edge physical network platform and discussed the challenges that must be solved first to effectively utilize the advantages of LV.

(2) The introduction of edge computing has greatly reduced the delay of task transmission in the IoT. However, when the number of tasks is at a high peak, processing multiple delay-sensitive tasks in the same edge server will cause additional waiting delays that reduce the quality of edge services and bring a poor user experience. Therefore, researchers began to work on solving the problem of service offloading that migrated tasks to relatively idle edge servers. Wang et al. [27] used Dynamic Voltage Scaling (DVS) to optimize the computing speed, transmit power, and offload rate of smart mobile devices, thereby reducing the energy consumption of smart mobile devices and the time delay in task execution. Yu et al. [28] considered that when multiple mobile users offload repetitive computing tasks to the edge of the network, they developed a cache-enhanced service offloading strategy based on sharing computing results to reduce the delay in task processing. As the subject of machine learning becomes more and more popular, researchers have also begun to study how to apply machine learning to edge IoT systems [29–31]. Liu et al. [28] grouped users according to priority. For grouped users of different priorities, Markov decision was adopted to design corresponding service

offloading strategies to reduce system costs, and the deep Q-network was used to train the model and determine the best service offloading strategy. Sangaiah et al. [32] proposed a method of using machine learning to protect the confidentiality of the user's location in response to the privacy leakage problem in the process of service offloading, which improves the security of users when using edge services.

The current research on edge server layout strategy based on the determined service offloading strategy are relatively small, but it is necessary to ensure the performance of edge services and the quality of real-time control by studying the layout of edge servers [15]. Therefore, this article proposes an edge server layout method for offloading services in the IoT, which aims to reduce the delay in the task transmission and the energy consumption of the edge servers and realize the overall load balance of the edge servers.

3. Edge Computing System Model Orienting Service Offloading

This section proposes an edge computing system model orienting service offloading and gives the corresponding computing formulas for the three optimization goals of delay, energy consumption, and load balancing in detail. The symbols and descriptions of this article are shown in Table 1.

3.1. Model Design. The edge computing model orienting services offloading in the IoT is shown in Figure 1.

In Figure 1, the edge computing model is divided into three levels: user layer, edge service layer, and cloud service layer. At the user level, smart devices transfer computationally intensive tasks to sensors. At the edge service layer, sensors $S = \{s_1, s_2, \ldots, s_N\}$ are deployed to collect the computationally intensive tasks of users in the coverage area. A small number of edge server $\mathrm{ES} = \{es_1, es_2, \ldots, es_M\}$ are deployed near some sensors to process the tasks collected by the sensors and perform corresponding operations based on the computing results to provide edge services for the devices. At the cloud service layer, the cloud platform handles tasks that require cloud services.

In this article, the delay, energy consumption, and load variance of different edge server layouts are used to judge the advantages and disadvantages of the edge server layout strategy. To facilitate comparison, for different edge server layout strategies, the sensor task transmission path determination rules are the same. After determining the mission transmission path of all sensors, three indicators of time delay, energy consumption, and load variance are computed. In addition, this article assumes that the CPU of the edge server is a single-core processor, and the configuration of each edge server is the same. Therefore, when there are tasks in the edge server being processed, the unprocessed sensor tasks will enter the waiting queue of the edge server waiting to be processed. At the same time, if a certain edge server has a lot of load tasks, the edge server can hand over the task that latest arrives at the edge server to an edge server that is relatively close and has less load.

Table 1: Symbols and corresponding descriptions.

Symbol	Description
ES	The collection of edge servers, $\mathrm{ES} = \{es_1, es_2, \ldots, es_M\}$
S	The collection of sensors, $S = \{s_1, s_2, \ldots, s_N\}$
Z	The number of rounds of the task in the waiting queue
ASL	The average delay of the edge server
TE	Total energy consumption
ALV	The average load variance of the edge server.
K	Number of chromosomes

3.2. Time Delay Model. The transmission and computing delays of sensor tasks include the task transmission delay of the task from the sensor to the edge server, the task processing delay of the edge server processing the task, the task waiting time of the task waiting in the edge server, and the return time of server returning the computing result to the sensor. Because the sensor can only cover a certain range, the edge server has two states: within the coverage area of the sensor and not within the coverage area of the sensor. When there is an edge server in the coverage area of the sensor, the sensor directly transmits the task to the edge server. When there is no edge server in the sensor coverage, the mission transmission path of the sensor needs to be further determined. Using a binary state variable Y_n^m to determine whether the m-th edge server es_m is within the coverage of the n-th sensor s_n,

$$Y_n^m = \begin{cases} 1, & es_m \text{ is with in the coverage of } s_n, \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

then the computing formula of the time for the sensor s_n to transmit the task to the server es_m is as follows:

$$AT_n = \left(1 - Y_n^m\right) \cdot sn_n^m \cdot \frac{ds_n}{\lambda}. \tag{2}$$

where ds_n is the data volume of the sensor s_n computing task, λ is the data transmission rate of the sensor to the sensor or edge server, and sn_n^m is the number of sensors passed by the sensor s_n to the edge server es_m.

The computing formula for the time spent by the edge server to compute the task of the sensor s_n is shown in formula (3), where c_n^m is the total number of clock cycles required for processing the task of the sensor s_n in the edge server es_m, and f_m is the main frequency of the edge server es_m.

$$BT_n = \frac{c_m^n}{fm'}. \tag{3}$$

Because the task scheduling in the server adopts the First Come First Service (FCFS) algorithm, the computing formula for the waiting time of the task of the sensor s_n in the server is as follows:

$$CT_n = \sum_{z=1}^{Z_n^m} ET_z^m - AT_n, \tag{4}$$

where Z_m^n is the number of rounds that the task of sensor s_n waits for computation in the edge server es_m, and ET_z^m is the time that the edge server es_m spends in the *z-th* round of task computation, but the task of edge server s_n does not always

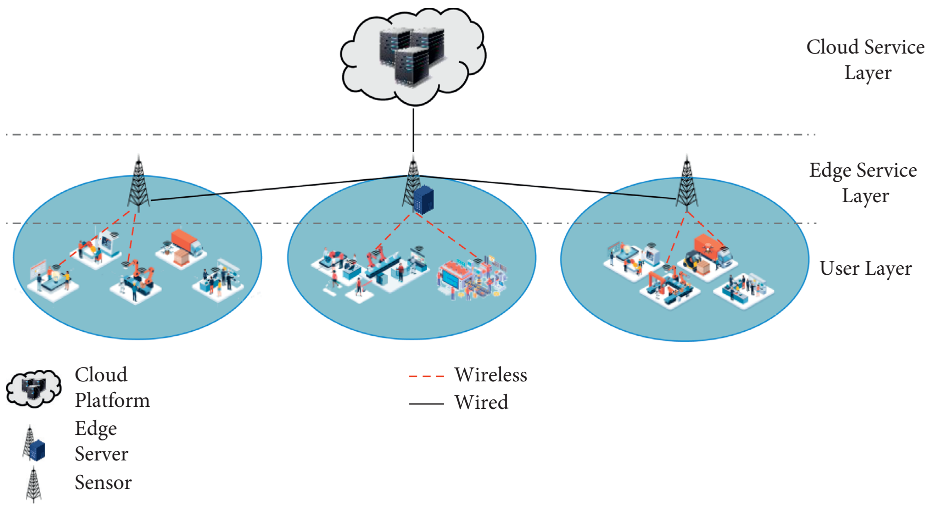


Figure 1: The edge computing model orienting services offloading in the IoT.

arrive at the edge server immediately, so the waiting time of this task needs subtracting the time delay spent in the task transmission process.

The computing formula for the time taken by the edge server to return the computing result to the sensor s_n is shown in formula (5), where ds_n' represents the task computing result of the sensor s_n.

$$DT_n = \left(1 - Y_n^m\right) \cdot sn_n^m \cdot \frac{ds_n'}{\lambda}, \tag{5}$$

where Y_n^m is a binary state variable, sn_n^m is the number of sensors passed by the sensor s_n to the edge server es_m.

Therefore, the task transmission delay of the task from the sensor to the edge server, the task processing delay of the edge server processing the task, the task waiting time of the task waiting in the edge server, and the result return time of the server returning the computing result to the sensor together constitute the task transmission delay and calculation delay of sensor s_n:

$$ST_n = AT_n + BT_n + CT_n + DT_n, \tag{6}$$

and then the formula for computing the average delay of all sensor tasks is as follows:

$$\text{AST} = \frac{1}{N}\sum_{n=1}^{N} ST_n. \tag{7}$$

3.3. Energy Consumption Model. Considering that sensors are always collecting environmental information and transmitting tasks to the edge server, the running time of the edge server is the key to computing energy consumption. The time computing formula for the service provided by the *m-th* edge server es_m is as follows:

$$\text{SPT}_m = \sum_{z=1}^{Z_m} ET_z^m, \tag{8}$$

where Z_m is the total number of rounds of the *m-th* edge server es_m computing task.

Defining ρ as the operating power of edge servers, the basic energy consumption to keep all edge servers running continuously is as follows:

$$AE = \sum_{m=1}^{M} \left(\text{SPT}_m \cdot \rho\right). \tag{9}$$

Defining σ as the power of edge server task processing, the energy consumption of the m-th edge server es_m task processing is as follows:

$$\text{VME}_m = \text{SPT}_m \cdot \sigma, \tag{10}$$

and then the operating energy consumption of all corresponding edge server computing tasks is as follows:

$$BE = \sum_{m=1}^{M} \text{VME}_m. \tag{11}$$

Defining τ as the power when a single edge server is idle, and the energy consumption of the *m-th* edge server es_m without task processing is as follows:

$$\text{EVE}_m = \left(\max_{m=1}^{M} \text{SPT}_m - \text{SPT}_m\right) \cdot \sigma, \tag{12}$$

where the SPT_m is the running time of the *m-th* edge server es_m, and σ is the power of edge server task processing. Then, the corresponding energy consumption when the edge server is idle:

$$CE = \sum_{m=1}^{M} \text{EVE}_m. \tag{13}$$

Therefore, the total energy consumption of the edge server is as follows:

$$TE = AE + BE + CE. \tag{14}$$

3.4. Load Model. Considering that load describes the number of computing tasks in all edge servers, it is more appropriate to use load variance to evaluate whether load balancing is achieved between edge servers. Defining a binary variable determines whether the m-th edge server es_m is being occupied:

$$EO_m = \begin{cases} 1, & \text{edge server } es_m \text{ is being occupied}, \\ 0, & \text{otherwise}. \end{cases} \quad (15)$$

The total number of edge servers occupied is as follows:

$$NE = \sum_{m=1}^{M} EO_m. \quad (16)$$

The load of the *m-th* edge server es_m is measured by the number of tasks processed in es_m, and the average load of the edge server is shown in formula (17), where TP_m represents the number of tasks processed in the *m-th* edge server es_m:

$$\text{ARU} = \frac{1}{NE} \sum_{m=1}^{M} TP_m. \quad (17)$$

Therefore, the load variance of the m-th edge server es_m is as follows:

$$LV_m = (\text{ARU} - TP_m)^2. \quad (18)$$

Finally, the average load variance of all occupied edge servers is as follows:

$$\text{ALV} = \frac{1}{NE} \sum_{m=1}^{M} LV_m. \quad (19)$$

3.5. Problem Definition. This article aims to minimize the average delay in formula (7), the total energy consumption in formula (14) and the variance of the average load in formula (19). The multiobjective optimization problem is defined as

$$\begin{aligned} &\min \text{AST}, \\ &\min TE, \\ &\min \text{ALV}, \\ &\text{s.t.} m \le n, \\ &\text{s.t.} \max_{m=1}^{M} ds_n \le \min_{m=1}^{M} wl_m, \end{aligned} \quad (20)$$

where wl_m represents the maximum processing task size of the *m-th* edge server es_m, and formula (21) guarantees that the total number of edge servers is less than the total number of sensors. In addition, formula (20) guarantees that the task size of a single sensor is less than the maximum processing task size of a single edge server.

4. Method of Edge Server Placement Orienting Service Offloading

In a scenario where hundreds of sensors are densely distributed, a certain number of edge server layout strategies are iteratively generated to minimize the delay in the service offloading process, minimize the energy consumption of the edge server, and balance the load of the edge server. In this case, a low-complexity suboptimal algorithm for solving the NP-hard problem is needed to find the optimal solution for the edge server layout. This article proposes an edge server layout strategy optimization method EPMOSO based on GA and PSO to iteratively optimize the edge server layout strategy.

4.1. Optimizing the Edge Server Layout Strategy Based on GA. GA is a commonly used heuristic algorithm for solving MINLP. It is an algorithm that simulates the law of survival of the fittest in nature to search for the optimal solution randomly. It only requires that the problem to be solved is computable. The GA process is divided into four steps: Initialization, Selection, Crossover, and Mutation. The details of each step of GA are as follows.

4.1.1. Determination of Initialization and Task Transmission Route. As GA simulates the law of survival of the fittest in nature, genes and chromosomes are important optimization objects for GA. In the process of using GA to iteratively optimize the layout strategy of edge servers, the layout position of a single edge server is regarded as a gene, and the genes corresponding to the layout positions of all edge servers together constitute a chromosome, let ESP_m denote the position of the m-th edge server and then $ESP = \{ESP_1, ESP_2, \ldots, ESP_M\}$ constitutes a chromosome. During the initialization, each chromosome randomly assigns geographic locations to each edge server, forming an initial set of edge server layout strategies for subsequent iterations.

Because this article studies the impact of edge server layout on edge service quality, to compare the advantages and disadvantages of different edge server layout strategies, the task transmission method is determined in advance. The sensor transmits the task to the edge server that is the closest physical distance to itself to reduce the task transmission delay. A load threshold is set for the edge server to achieve load balancing of the edge server as much as possible. When the load of the edge server to which the sensor is to be transmitted exceeds the load threshold, the sensor transmits the task to the next edge server with the closest physical distance excluding this edge server. The task sensor path confirmation algorithm is shown in Algorithm 1.

4.1.2. Selecting Fitness Function. In GA, the fitness function is used to measure the adaptability of the chromosome. The fitness function is based on the average delay of the task transmission process shown in formula (7), and the total energy consumption of the edge server shown in formula (14). The average load variance of the edge server shown in formula (19) is used to compute the chromosome fitness. Tower the transmission delay, the lower the total energy consumption of the edge server and the lower the average load variance of the edge server, the better the server layout strategy of the corresponding edge server. Therefore, when solving the problem of edge server layout, the lower the fitness of a chromosome, the more it indicates that the chromosome has strong adaptability in the chromosomes of this iteration. The fitness function is defined as formulas

```
     Inputs: Sensor $S$, Edge server $ES$, Task threshold of ES
     Output: Determined task transmission path
(1)    for $n = 1$ to $N$ do
(2)      Find the nearest edge server $es_m$
(3)      if the number of task of edge server $es_m \le$ task threshold of ES then
(4)        Delete $es_m$ from ES
(5)        Go to step 2
(6)      else
(7)        Confirm that the sensor sn will transmit the task to the edge server $es_m$
(8)        Number of tasks of edge server $es_m + 1$
(9)      end if
(10)   end for
(11) return Determined task transmission path
```

Algorithm 1: Task transmission path confirmation algorithm.

(23)–(26). Formula (23) represents the fitness function of transmission delay, formula (24) represents the fitness function of edge server energy consumption, formula (25) represents the fitness function of edge server load balancing, and formula (26) represents the comprehensive fitness function of the *k-th* chromosome, where w_d, w_e, and w_l are the weights of delay, energy consumption, and load balancing fitness, respectively.

After computing the fitness, GA performs selection operations based on the adaptive capacity of each chromosome. In the selection operation, the commonly used selection algorithms include Roulette Wheel and Tournament. The roulette algorithm puts all chromosomes into a wheel for selection, and the probability of each chromosome being selected is proportional to its fitness. Therefore, the roulette algorithm is more suitable for the problem of maximum optimization. However, the championship algorithm not only requires low computing resources but also does not need to modify the code itself when applied to the minimum optimization problem. Therefore, this article uses the tournament algorithm as the selection algorithm. The tournament algorithm will select the few chromosomes with the strongest adaptability, that is, the lower adaptability, to copy directly for the next iteration.

$$\begin{cases} DF_k = \dfrac{\mathrm{ASK}_k}{\sum_{k=1}^{K} \mathrm{ASK}_k}, \\[2ex] EF_k = \dfrac{TE_k}{\sum_{k=1}^{K} TE_k}, \\[2ex] LF_k = \dfrac{\mathrm{ALV}_k}{\sum_{k=1}^{K} \mathrm{ALV}_k}, \end{cases} \tag{21}$$

$$AF_k = w_d \cdot DF + w_e \cdot EF + w_l \cdot LF_k. \tag{22}$$

4.1.3. Crossover and Mutation. In order to increase the diversity of chromosomes and extend the search range of strategies to generate better edge server layout strategies and search for global optimal solutions and avoid falling into local optimal solutions, GA introduces crossover and mutation operations. Selection operations retain a certain number of ancestral chromosomes, while crossover and mutation operations are used to generate a certain number of descendant chromosomes.

In the process of crossover, two ancestral chromosomes are randomly selected to exchange several corresponding genes, thereby generating two new offspring chromosomes. When dealing with the multiobjective optimization problem, the probability of crossover is designed for the crossover operation, and each gene is based on the possibility of crossover to exchange, that is, if the gene needs to be crossed, the corresponding edge server positions of the two ancestral chromosomes will be exchanged. If other edge servers of the exchanged chromosomes have been deployed in the position to be exchanged, the edge server will be reset to a new location. An example of the crossover operation is shown in Figure 2(a).

In the process of mutation, two ancestral chromosomes are randomly selected, and each gene on the two chromosomes mutates according to the mutation probability, thereby generating a new descendant chromosome. When dealing with the multiobjective optimization problem, each gene on the chromosome is mutated according to the mutation probability, that is, if the gene needs to be mutated, the edge server will be randomly deployed in a random location, if other edge servers have been deployed in this new location, then a random algorithm is used to reallocate the location until no edge server is deployed in the location. An example of the mutation operation is shown in Figure 2(b).

Due to the use of the PSO algorithm to optimize the performance of the GA algorithm, to facilitate the implementation of the subsequent PSO algorithm, in addition to the basic steps of the GA, the chromosome corresponding to the historical minimum fitness and the historical minimum fitness during the iteration process of the numbered chromosome will also be recorded. The corresponding chromosomes are used for PSO algorithm iteration, and these chromosomes are updated after each round of GA iteration. Algorithm 2 describes the specific process of Genetic Algorithm iterative optimization of edge server layout strategy.

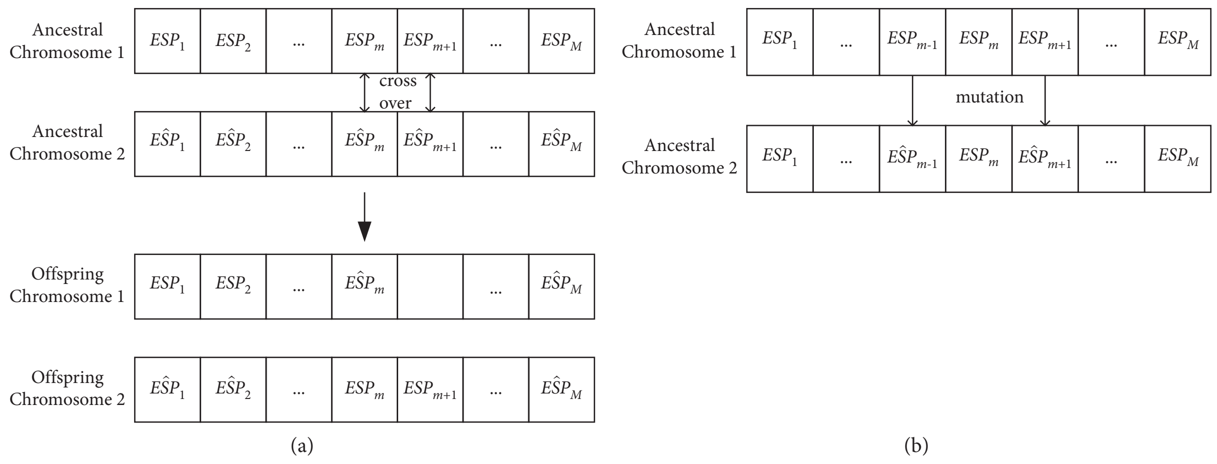

Ancestral Chromosome 1
ESP_1 ESP_2 ... ESP_m ESP_{m+1} ... ESP_M
cross over
Ancestral Chromosome 2
$\hat{ESP}_1$ $\hat{ESP}_2$ ... $\hat{ESP}_m$ $\hat{ESP}_{m+1}$ ... $\hat{ESP}_M$
Offspring Chromosome 1
ESP_1 ESP_2 ... $\hat{ESP}_m$ ... $\hat{ESP}_M$
Offspring Chromosome 2
$\hat{ESP}_1$ $\hat{ESP}_2$ ... ESP_m ESP_{m+1} ... $\hat{ESP}_M$
(a)
Ancestral Chromosome 1
ESP_1 ... ESP_{m-1} ESP_m ESP_{m+1} ... ESP_M
mutation
Ancestral Chromosome 2
ESP_1 ... $\hat{ESP}_{m-1}$ ESP_m $\hat{ESP}_{m+1}$ ... ESP_M
(b)

Figure 2: Chromosome crossover and mutation. (a) Chromosome crossover operation. (b) Chromosome mutation operation.

4.2. Optimizing the Edge Server Layout Strategy Based on PSO. PSO is also a heuristic algorithm commonly used to solve MINLP. It is an intelligent population optimization algorithm based on social information sharing. The algorithm originated from the study of the information sharing behavior of bird flocks during predation. The purpose of individuals in a flock is to search for food, but the individual does not know the specific location of the food. When an individual in the flock finds food, it will share the location of food with the flock, and then, other individuals will move in that direction. With the continuous sharing of information, all individuals in the flock can find food. PSO does not have the crossover and mutation operations of GA and has the advantages of high accuracy and fast convergence.

Particles are the optimization objects of the PSO algorithm. Because this article uses PSO to optimize the performance of GA, the particles of the PSO algorithm are equivalent to the genes of GA. But to facilitate the realization of the PSO algorithm, a speed attribute is added to the GA gene. Therefore, the particle (gene) has two attributes: position and speed. The position attribute of the particle represents the layout position of a single edge server in the current state, and the speed attribute of the particle controls the direction and distance of the layout position of the edge server. The process of PSO includes two steps: initialization and particle attribute update:

(1) Initialization: considering that PSO is used to optimize the convergence speed of GA, the PSO initialization part in this article cancels the initialization of the position attributes of each particle and uses the chromosome generated by GA iteration as input to initialize the speed of each particle.

(2) Particle attribute update: the particles are deployed separately in the scene, and the optimal solution found is recorded as the individual optimal position, and the optimal solution found in the population is recorded as the global optimal position. To avoid the result of the population falling into the local optimal solution, it is necessary to adjust two attributes of speed and position according to the global optimal position when each particle adjusts the two attributes of speed and position.

$$V_{t+1}^{k,m} = V_t^{k,m} + c_1 \cdot \left(\text{pbest}_t^m - X_t^{k,m}\right) + c_2 \cdot \left(\text{gbest}^m - X_t^{k,m}\right), \quad (23)$$

$$X_{t+1}^{k,m} = X_t^{k,m} + V_{t+1}^{k,m}. \quad (24)$$

The velocity and position of the particles are iterated according to formulas (23) and (24), where $V_t^{k,m}$ represents the velocity of the m-th particle of the k-th chromosome at the t-th iteration, and $X_t^{k,m}$ represents the position of the m-th particle of k-th chromosome. $c1$ and $c2$ are acceleration constants, generally set to 0.5. $pbest_t^m$ represents the best position of the m-th particle in the t-th iteration, and $gbest^m$ represents the best position of the m-th particle in the historical iteration.

In the process of using PSO to iteratively optimize the edge server layout strategy, each gene of the chromosome in the iteration process adjusts its position based on the chromosome the historical minimum fitness recorded by GA and the historical minimum fitness during the iteration. Algorithm 3 describes the specific process of using PSO to optimize GA iteratively optimized chromosomes.

4.3. Selecting the Optimal Edge Server Layout Strategy Based on SAW. A set of optimized edge server layout strategies optimized by GA and PSO iteratively; the SAW decision-making method is used to determine the final edge server layout strategy. First, the three indicators of delay, energy consumption, and load of the edge server layout strategy were standardized:

```
     **Inputs**: $K$ chromosomes, Number of iterations $T$, variable $t$, the number of chromosomes to operate k
     **Output**: $K$ chromosomes after GA iteration optimization
(1)    for $t = 0$ to $T$ do
(2)      if $t == 0$ then
(3)        Initialize $K$ chromosomes
(4)      else
(5)        Compute the fitness of K chromosomes and select $k$ chromosomes with lower fitness
(6)        Record the chromosome with the lowest fitness
(7)        Select two chromosomes randomly and perform crossover operations
(8)        Select two chromosomes randomly and perform mutation operations
(9)      end if
(10)   end for
(11) return $K$ chromosomes
```

ALGORITHM 2: Optimizing edge server layout strategy based on GA.

```
     **Inputs**: $K$ chromosomes that have been iteratively optimized by GA, Number of iterations $T$, Variable $t$
     **Output**: $K$ chromosomes optimized by PSO
(1)    for $t = 0$ to $T$ do
(2)      if $t == 0$ then
(3)        Initialize the velocity of each particle of K chromosomes
(4)      **else**
(5)        Adjust the position of each particle according to formulas (23) and (24)
(6)      end if
(7)    end for
(8)    **return** $K$ chromosomes
```

ALGORITHM 3: Edge server layout strategy based on PSO.

$$DU = \begin{cases} \dfrac{\mathrm{AST}_{\max} - \mathrm{AST}}{\mathrm{AST}_{\max} - \mathrm{AST}_{\min}}, & \mathrm{AST}_{\max} \neq \mathrm{AST}_{\min}, \\ 1, & \mathrm{AST}_{\max} = \mathrm{AST}_{\min}, \end{cases}$$
$$EU = \begin{cases} \dfrac{TE_{\max} - TE}{TE_{\max} - TE_{\min}}, & TE_{\max} \neq TE_{\min}, \\ 1, & TE_{\max} = TE_{\min}, \end{cases} \quad (25)$$
$$LU = \begin{cases} \dfrac{\mathrm{ALV}_{\max} - \mathrm{ALV}}{\mathrm{ALV}_{\max} - \mathrm{ALV}_{\min}}, & \mathrm{ALV}_{\max} \neq \mathrm{ALV}_{\min}, \\ 1, & \mathrm{ALV}_{\max} = \mathrm{ALV}_{\min}, \end{cases}$$

where $AST_{\max}$, $AST_{\min}$, $TE_{\max}$, $TE_{\min}$, $ALV_{\max}$, and $ALV_{\min}$, respectively, represent the highest and lowest transmission delay of the chromosomes optimized iteratively, the energy consumption of the edge server, and the variance of the overall load balance of the edge server. In the standardization operation, there are two ways to compute the numerator: the maximum value of the optimization index minus the current value of the optimization index and the current value of the optimization index minus the minimum value of the optimization index. Taking the delay of the transmission into account, the total energy consumption of the edge server and the variance of the average load of the edge server are both minimum optimization problems. Therefore, in the standardization operation, the current value of the optimization index is subtracted from the maximum value of the optimization index.

After standardizing each index, the utility value of the corresponding chromosome is computed according to the corresponding weight of delay, energy consumption, and load balance, as shown in formula (26), where DU, EU, and LU are standardized values of delay, energy consumption, and the variance of load, respectively, and w_d, w_e, and w_l are the weights corresponding to the delay, energy consumption, and variance of load, respectively. The utility value is used to determine whether the edge server layout strategy corresponding to the chromosome is the optimal edge server layout strategy. After computing the utility value, the edge server layout strategy with the highest utility value is selected as the final edge server layout strategy:

$$AU = w_d \cdot DU + w_d \cdot EU + w_l \cdot LU. \quad (26)$$

4.4. Summary of the Method. Both GA and PSO are heuristic algorithms commonly used to solve MINLP. The GA algorithm simulates the natural phenomenon of retaining excellent genes and eliminating genes that are not suitable for the environment during the population iteration process, thereby solving the target problem. The advantage of GA is that it can better find the global optimal solution, but its convergence speed is slow, and a satisfactory solution can only be obtained when the number of iterations is large. The advantage of the PSO algorithm is that the algorithm is simple, and the convergence speed is fast, but it can often only find the target solution in a limited search area, so it is easy to fall into the local optimal solution instead of the global optimal solution [33]. Therefore, in this article, to integrate the advantages of GA and PSO, use PSO to optimize GA and propose an edge server layout strategy research method for offloading services in IoT, EPMOSO, which uses GA and PSO to iteratively optimize a set of excellent edge server layout strategy, and finally use SAW method to determine the final edge server layout strategy. An example of EPMOSO is shown in Figure 3.

Algorithm 4 describes the core process of EPMOSO. Algorithm 3, taking the sensor set, edge server set, and iteration number as input, first randomly allocates the initial test position for the edge server and initialize the chromosomes, use the set of chromosomes as the input of GA, and perform coarse-grained optimization. Then, selection, crossover, and mutation are performed; the chromosome of this iteration is used as the input of PSO, and the position of each gene is updated. After the fine-grained optimization of PSO, this iteration ends. Until the number of iterations is full and output a set of optimized edge server layout strategies. For this group of edge server layout strategies, compute its utility value and use the SAW method to determine the final edge server layout strategy.

5. Experiment Analysis

In this section, the effectiveness of EPMOSO is evaluated by analyzing the results of comparative experiments. First, introduce the experimental configuration of the experiment in this article and introduce the content of the algorithm compared with EPMOSO. Then, according to the experimental results, the pros and cons of the EPMOSO and other methods proposed in this article are analyzed from different angles.

5.1. Experimental Configuration. In comparison experiments, the effectiveness and efficiency of EPMOSO were compared with the other algorithm in the case of different edge server sizes. The parameter settings of this experiment are shown in Table 2, and the two comparison algorithms in the experiment are shown as follows:

(1) Genetic Algorithm [33]: the genetic algorithm and EPMOSO are used separately to compare the convergence trends of the two methods to evaluate the convergence speed of EPMOSO, to highlight that the EPMOSO method still has a good convergence speed when the global optimal solution can be found.

(2) Particle population algorithm [34] the particle population algorithm PSO and EPMOSO are used separately to verify whether the edge server layout strategy iterated by EPMOSO is the global optimal solution and highlight the advantages of EPMOSO over PSO in solving the global optimal solution.

5.2. Comparative Analysis. In the comparative analysis, the effectiveness of EPMOSO, GA, and PSO is analyzed from four aspects: the delay of the transmission, the total energy consumption of the edge server, the average load variance, and the utility value of the edge server. In addition, the convergence rate of the three methods is evaluated by analyzing the convergence curves of the indicators of the three algorithms in the iterative process. Finally, under different edge server scales, the optimal solution selected by SAW selection iteration is presented in Figure 10.

5.2.1. Comparison of Delay. In the experiment of this article, all edge servers have the same configuration, so the task processing speed of each edge server is the same. Under the same edge server scale, the task processing time of the edge server during the iteration of EPMOSO, GA, and PSO is always the same. EPMOSO, GA, and PSO mainly optimize the transmission time of the task and the waiting time of the task in the edge server. In addition, because tasks have more edge servers that are closer to each other as task transmission options, and there is no need to migrate tasks to edge servers that are farther away, the task transmission time will decrease as the number of edge servers increases. In addition, due to the increase in the number of edge servers, the number of tasks gathered on the same edge server is reduced, so the waiting time of tasks in the edge server is also reduced. In Figure 4, it can be seen from the image that when the edge server scale is 15, 20, 25, and 30, compared with GA and PSO, EPMOSO has a stronger ability to optimize latency.

5.2.2. Comparison of Energy Consumption. In the experiment, considering that the energy consumption of a single edge server is related to the number of tasks gathered in the edge server and the processing time of these tasks, as the number of edge servers increases, the number of tasks gathered in a single edge server is relatively reduced. The energy consumption of the server is also relatively reduced. Although the energy consumption of a single edge server is relatively reduced, the increase in the number of edge servers leads to an increase in the total energy consumption of edge servers. Figure 5 compares the total energy consumption of edge servers under different edge server sizes. It can be seen from the figure that when the edge server scale is 15, 20, 25, and 30, compared with GA and PSO, EPMOSO has relatively small advantages in energy consumption optimization.

5.2.3. Comparison of Load Variance. In the experiment, the task load threshold is set for each edge server. Therefore, before the algorithm is optimized, all edge servers have basically realized load balancing, but with the continuous optimization of the algorithm, the layout of edge servers tends to be reasonable, then the load variance of the edge server will still achieve a certain optimization. Figure 6 presents the edge server load variances optimized by

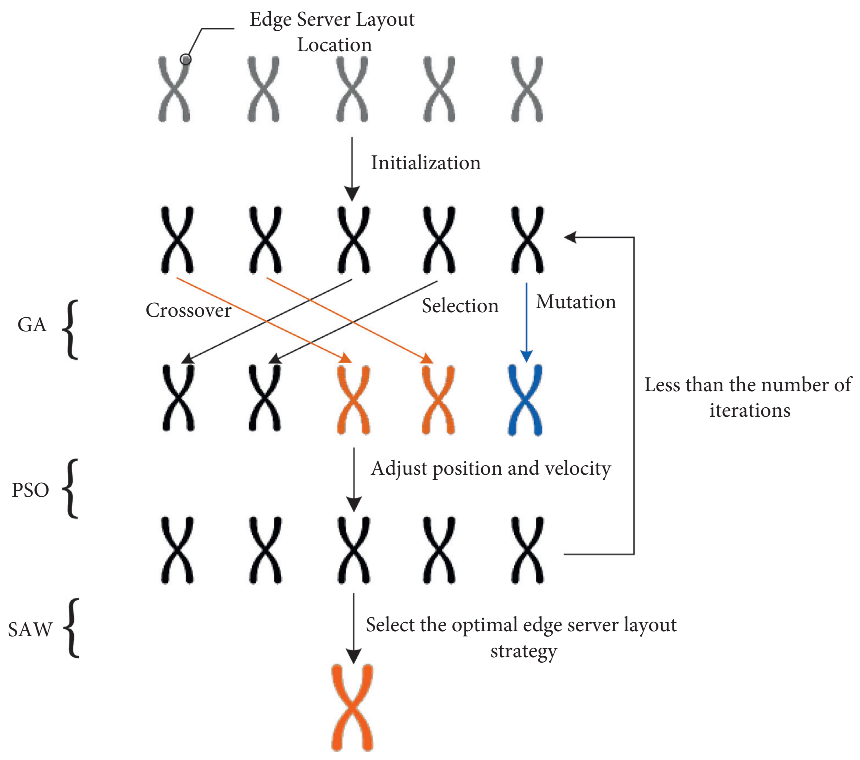

Edge Server Layout Location
Initialization
GA
Crossover
Selection
Mutation
Less than the number of iterations
PSO
Adjust position and velocity
SAW
Select the optimal edge server layout strategy

Figure 3: Example of EPMOSO.

```
    Inputs: Sensor collection $S$, Edge server collection $ES$, Number of iterations $T$, Variable $t$
    Output: Determined final edge server layout strategy
(1) Randomly set the location of the edge server and initialize the chromosome
(2)   Initialize the velocity of each gene of each chromosome
(3)   for $t = 0$ to $T$ do
(4)     Execute GA algorithm according to Algorithm 2
(5)     Execute PSO algorithm according to Algorithm 3
(6)   end for
(7)   Compute the utility value of $K$ chromosomes
(8)   Use SAW algorithm to determine the final edge server layout strategy
(9) return Best Edge Server Layout Strategy
```

Algorithm 4: Summary of EPMOSO method.

Table 2: Experimental parameters.

Parameter	Value
Number of sensors N	200
Number of edge servers M	15, 20, 25, 30
Threshold of the number of edge server tasks	18, 14, 12, 8
Sensor task size	[30–100] MB
Task transfer rate	10 MB/s
Idle power of edge server	100 W
Operating power of edge server	300 W
Number of chromosomes	10
Number of iterations	50
Number of chromosomes selected	6
Number of chromosome crossovers	2
Probability of chromosome crossovers	0.6
Number of chromosome mutations	2
Probability of chromosome mutations	0.8

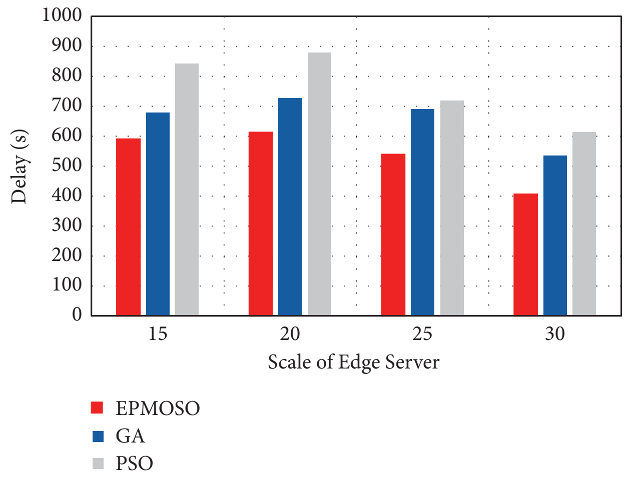


FIGURE 4: Comparison of delay under different edge server scales.

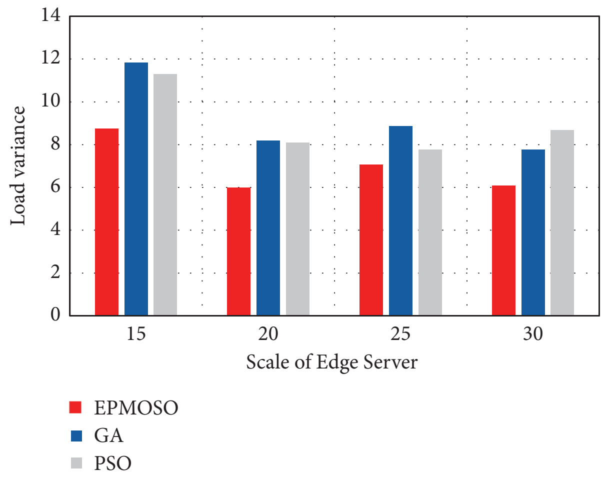


FIGURE 6: Comparison of load variance under different edge server scales.

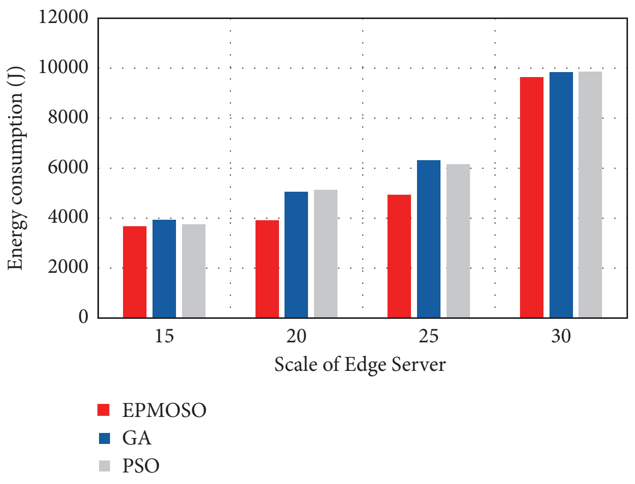


FIGURE 5: Comparison of energy consumption under different edge server scales.

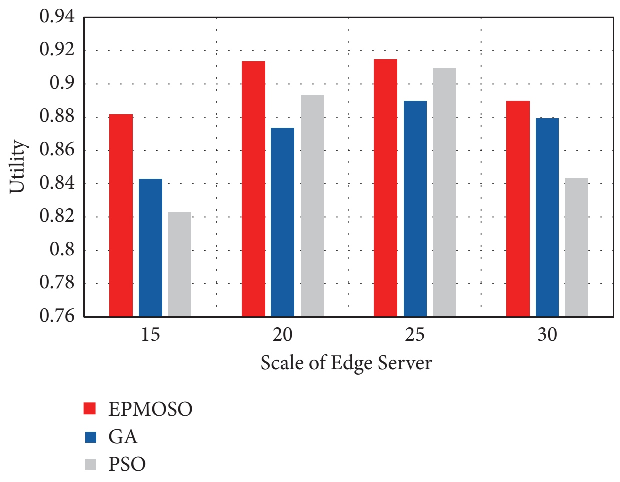


FIGURE 7: Comparison of utility under different edge server scales. (a) Edge server scale: 15. (b) Edge server scale: 20. (c) Edge server scale: 25. (d) Edge server scale: 30.

EPMOSO, GA, and PSO under different edge server sizes. It can be seen from the image that when the edge server size is 15, 20, 25, and 30, EPMOSO still shows better optimization capabilities.

5.2.4. Comparison of Utility. In this experiment, the SAW method is used to determine the final edge server layout strategy. The SAW method first computes the utility value of the chromosome. The higher the utility value, the better the edge server layout strategy corresponding to the chromosome. Therefore, in the continuous iterative optimization process, the utility value of the chromosome will be higher and higher, indicating that the solution searched by the optimization algorithm is getting closer and closer to the global optimal solution. Figure 7 presents the utility value of the edge server layout strategy determined by SAW and iterative optimization of EPMOSO, GA, and PSO under different edge server scales. Obviously, when the edge server scale is 15, 20, 25, and 30, the edge server layout strategy optimized by EPMOSO has a higher utility value. Therefore, compared with GA and PSO, using EPMOSO for iterative optimization can obtain a better edge server layout strategy.

5.2.5. Comparison of Convergence Speed. In the experiment, considering that if there are too many iterations, EPMOSO, GA, and PSO will gradually tend to converge, and it is impossible to directly determine the convergence speed of EPMOSO through comparison. Therefore, the comparison experiment of EPMOSO, GA, and PSO iteratively optimizes the edge server layout strategy is set to 50 times. After EPMOSO, GA, and PSO are initialized, they each iterate 50 times to iteratively optimize a set of edge server layout strategies. Figures 7–9 compare the process of EPMOSO, GA, and PSO to optimize the delay of the task transmission process, the total energy consumption of the edge servers, and the load variance of the edge serves.

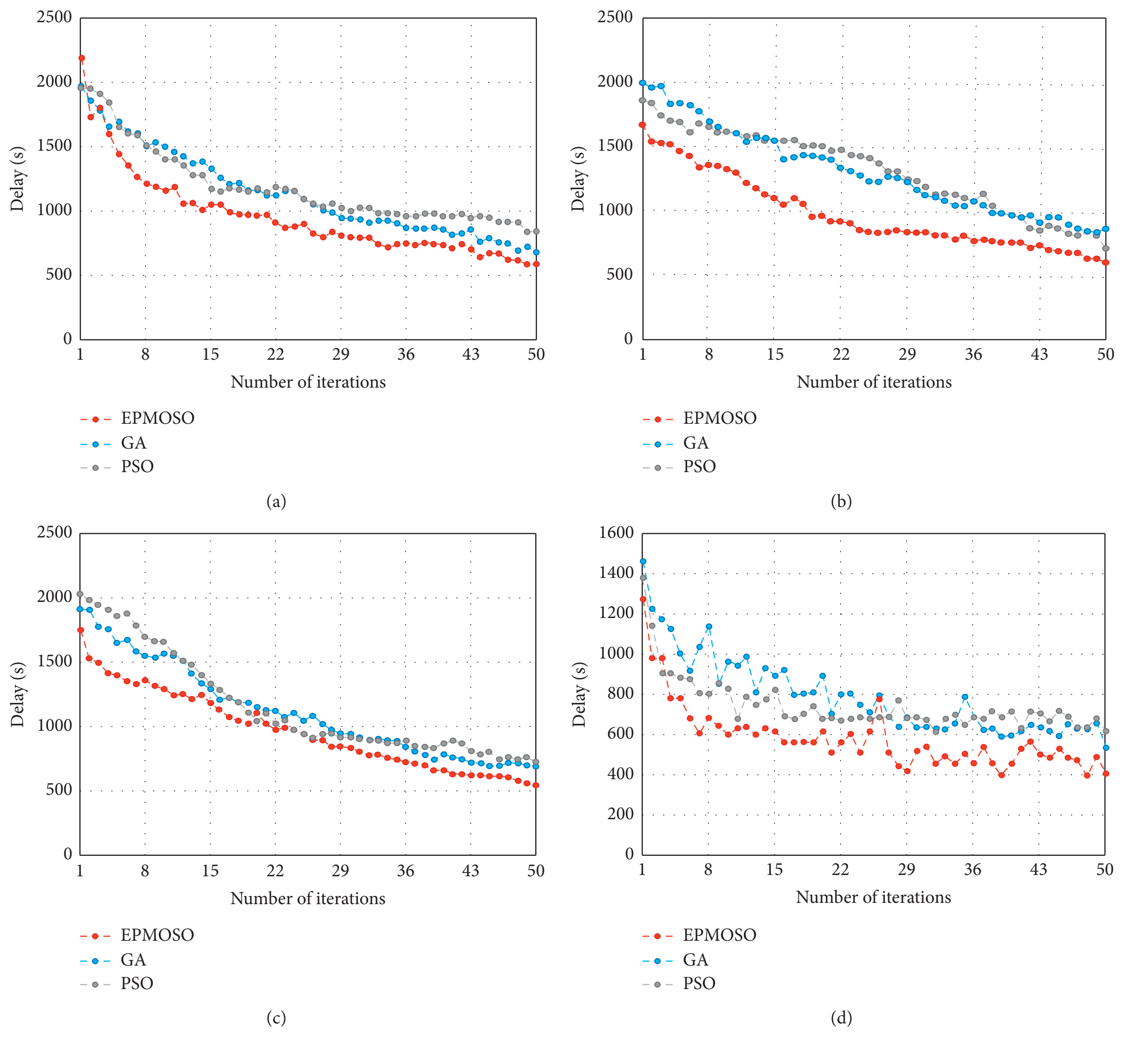
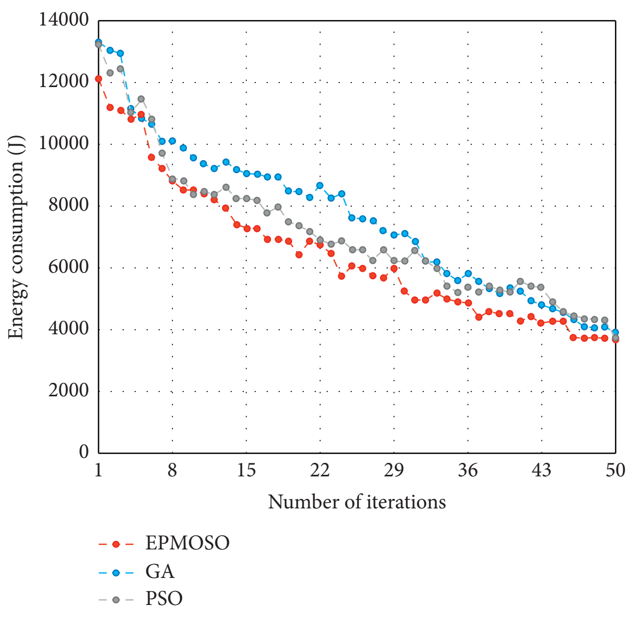


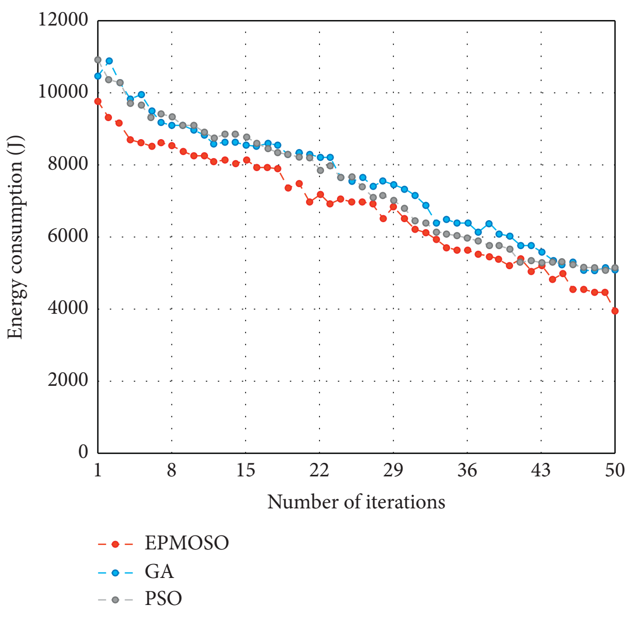
Figure 8: Comparison of delay convergence rate under different edge server scales. (a) Edge server scale: 15. (b) Edge server scale: 20. (c) Edge server scale: 25. (d) Edge server scale: 30.

In Figure 8, Figures 8(a)–8(d) compare the changes of delay in the iterative process when the edge server size is 15, 20, 25, and 30. In Figures 8(a)–8(d), EPMOSO showed a faster convergence rate in the first 20 iterations and a lower delay after optimization. In the next 30 iterations, the edge server layout strategy in the iteration is getting closer and closer to the global optimal solution, so the convergence speed of the three algorithms slows down. Therefore, compared with GA and PSO, EPMOSO not only has a better effect in optimizing the delay of the transmission process but also has a better convergence speed. Therefore, EPMOSO is effective for optimizing the time delay of the task transmission process.

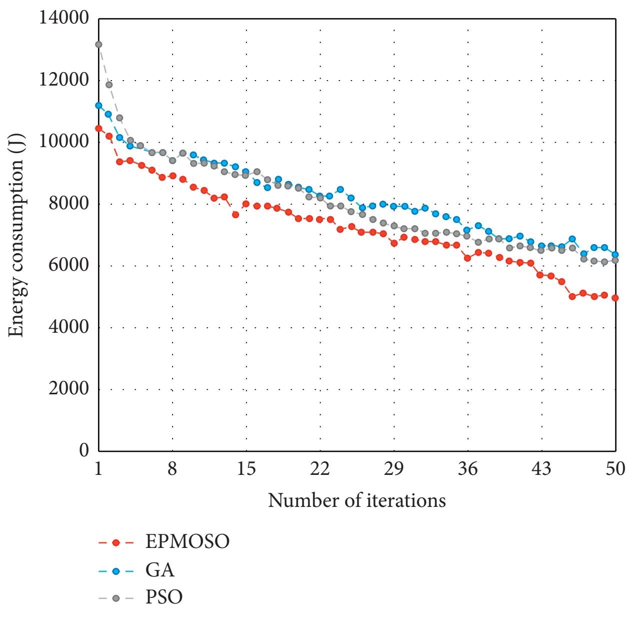
In Figure 9, Figures 9(a)–9(d) compare the changes in energy consumption during the iteration. When the scale of edge server is 15, EPMOSO spend lower energy initially than GA and PSO, but as the number of iterations increases, the energy consumption of EPMOSO and PSO is similar, even when the number of iterations reaches 50, the energy consumption of three is almost same. When the scale of edge server is 20, the performance of the three is almost the same, but EPMOSOS has a slight advantage throughout the iteration process, the more the number of iterations, the more obvious its advantage. When the scale of edge server is 25, the same conclusion can be drawn as when the scale is 20. When the scale of edge server is 30, EPMOSO's performance is slightly worse than the other two algorithms initially; when the number of iterations reaches 10, its energy consumption is slightly better than the other two algorithms but fluctuates up and down. Although compared with GA and PSO, EPMOSO's advantage in convergence speed is not obvious, but EPMOSO's optimization of the total energy consumption of edge servers is effective.



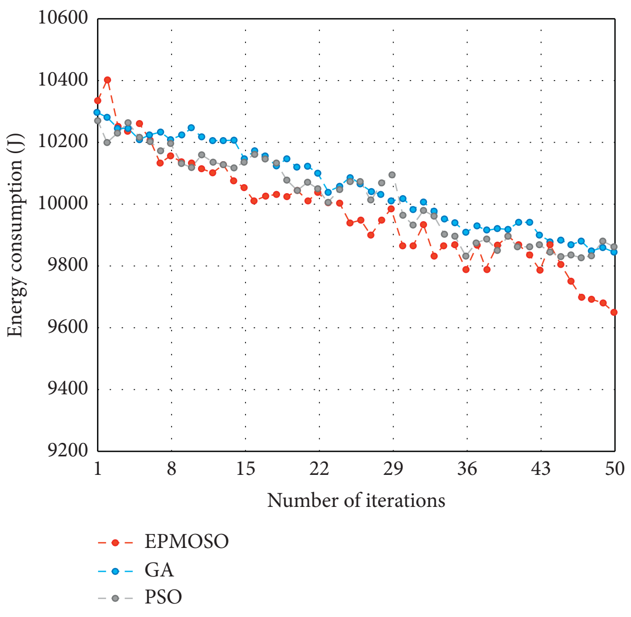
(a)



(b)



(c)



(d)

Figure 9: Comparison of the convergence rate of energy consumption under different edge server scales. (a) Edge server scale: 15. (b) Edge server scale: 20. (c) Edge server scale: 25. (d) Edge server scale: 30.

In Figure 10, Figures 10(a)–10(d) compare the difference in load variance during the iteration. When the scale of edge server is 15, EPMOSO showed a faster convergence rate in the first 35 iterations and a lower load variance, but the convergence rate of the three afterward is almost the same. When the edge server scale is 20, the convergence speed of the three is not much different, but EPMOSO is slightly better than the other two. When the scale of edge server is 25, the same conclusion can be drawn as when the scale is 20. When the scale of edge server is 30, the convergence speed of EPM and PSO in the first 20 iterations are basically the same, but it is better than the other two afterward. Considering that this article has set task thresholds for all edge servers, it is reasonable that the optimization effect of edge server load variance is not obvious. However, it can be seen from the figure that EPMOSO can still optimize the load variance of edge servers to a certain extent.

In summary, EPMOSO can optimize the delay of the task transmission process, the total energy consumption of the edge server, and the load variance of the edge server. In the

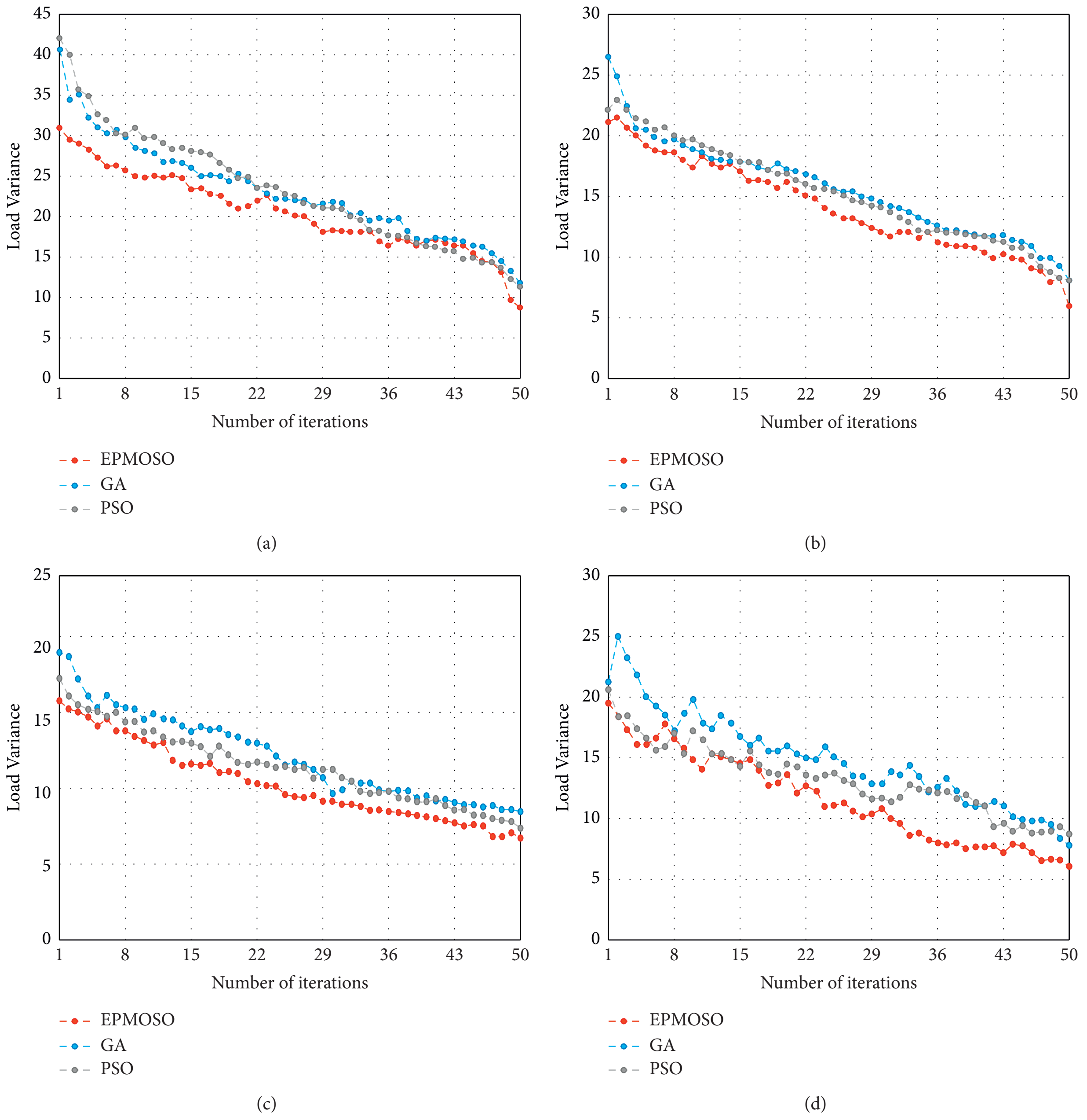


Figure 10: Comparison of the convergence rate of load variance under different edge server scales. (a) Edge server scale: 15. (b) Edge server scale: 20. (c) Edge server scale: 25. (d) Edge server scale: 30.

comparison between EPMOSO and GA, although GA also can find the global optimal solution, when the number of iterations is 50, GA obviously cannot achieve better convergence, and EPMOSO can not only find the global optimal solution but also achieve relatively better convergence. In comparing EPMOSO and PSO, PSO converges faster in the early stage of the iterative process. However, PSO is more likely to fall into a local optimal solution, so the convergence speed of PSO slows down in the later stage of the iterative process, and the optimization effect becomes worse. While EPMOSO maintains a good convergence rate, the optimization effect is still good.

5.2.6. Optimal Layout Strategy. The experiment in this article uses GA and PSO to iteratively optimize a set of excellent edge server layout strategies, and the SAW method determines the final edge server layout strategy. The optimal edge server layout strategy under different edge server scales is given in Figure 11. In theory, the layout of the edge server should be symmetrical and regular but considering that the task size of the sensor in this experiment is only a fixed range, and the task size of different sensors is set randomly; therefore, the layout of the edge server of the experimental results is asymmetric and irregular.

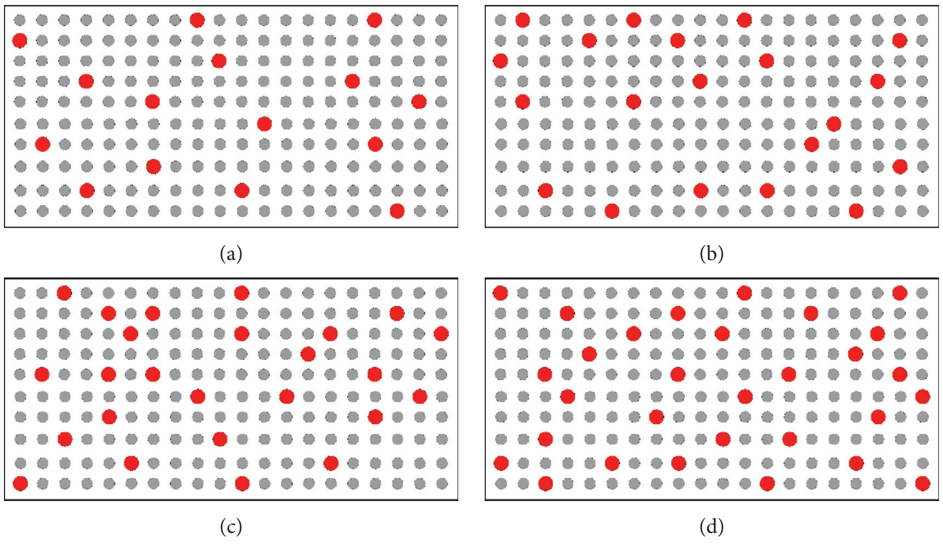


Figure 11: Edge server layout under different edge server scales. (a) Edge server scale: 15. (b) Edge server scale: 20. (c) Edge server scale: 25. (d) Edge server scale: 30.

6. Conclusion

Aiming at the problem of poor edge service quality and poor real-time control effect under a relatively small number of edge server scales, a research method of edge server layout strategy for offloading services in IoT is proposed to provide services with lower latency while reducing the edge server energy consumption and ensuring the stability of the edge server system. This method quantifies the above problem as a multiobjective optimization problem, which aims to reduce the time delay of the task transmission process and the energy consumption of the edge server while realizing the overall load balance of the edge server and proposes EPMOSO, which is a research method of edge server layout strategy for offloading services in IoT. This method first uses GA and PSO to iteratively optimize a set of excellent edge server layout strategies and then uses the SAW method to determine the optimal edge server layout strategy. The results of comparative experiments show that the EPMOSO method proposed in this article has the advantage of a better convergence speed when the global optimal solution can be found.

Data Availability

The raw data required to reproduce these findings cannot be shared at this time as the data also form part of an ongoing study.

Conflicts of Interest

The author declares no conflicts of interest.

References

[1] Q. Wu, K. He, and X. Chen, "Personalized federated learning for intelligent IoT applications: a cloud-edge based framework," *IEEE Open Journal of the Computer Society*, vol. 1, pp. 35–44, 2020.

[2] X. Xu, Z. Fang, J. Zhang et al., "Edge content caching with deep spatiotemporal residual network for IoV in smart city," *ACM Transactions on Sensor Networks*, vol. 17, no. 3, pp. 1–33, 2021.

[3] W. H. Hassan, "Current research on Internet of Things (IoT) security: a survey," *Computer Networks*, vol. 148, pp. 283–294, 2019.

[4] L. D. Xu, Y. Lu, and L. Li, "Embedding blockchain technology into IoT for security: a survey," *IEEE Internet of Things Journal*, vol. 8, no. 13, Article ID 10452, 2021.

[5] H. Elazhary, "Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: d," *Journal of Network and Computer Applications*, vol. 128, pp. 105–140, 2019.

[6] L. Kong, M. K. Khan, F. Wu, G. Chen, and P. Zeng, "Millimeter-wave wireless communications for IoT-cloud supported autonomous vehicles: overview, design, and challenges," *IEEE Communications Magazine*, vol. 55, no. 1, pp. 62–68, 2017.

[7] T. Wang, G. Zhang, A. Liu, B. Md Zakirul Alam, and Q. Jin, "A secure IoT service architecture with an efficient balance dynamics based on cloud and edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4831–4843, 2018.

[8] X. Xu, Q. Huang, Y. Zhang, S. Li, L. Qi, and W. Dou, "An LSH-based offloading method for IoMT services in integrated cloud-edge environment," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 16, no. 3s, pp. 1–19, 2021.

[9] M. Jia, Z. Yin, D. Li, and Q. Guo, "Toward improved offloading efficiency of data transmission in the IoT-cloud by leveraging secure truncating OFDM," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4252–4261, 2018.

[10] T. Wang, Y. Lu, J. Wang, H. N. Dai, X. Zheng, and W. Jia, "EIHDP: edge-intelligent hierarchical dynamic pricing based on cloud-edge-client collaboration for IoT systems," *IEEE Transactions on Computers*, vol. 70, 2021.

[11] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in IoT edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1133–1146, 2020.

[12] L. Lei, H. Xu, X. Xiong, K. Zheng, and W. Xiang, "Joint computation offloading and multiuser scheduling using approximate dynamic programming in NB-IoT edge computing system," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5345–5362, 2019.

[13] H. Tian, X. Xu, T. Lin et al., "DIMA: distributed cooperative microservice caching for Internet of Things in edge computing by deep reinforcement learning," *World Wide Web*, 2021.

[14] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *Journal of Parallel and Distributed Computing*, vol. 127, pp. 160–168, 2019.

[15] G. Cui, Q. He, F. Chen, H. Jin, and Y. Yang, "Trading off between user coverage and network robustness for edge server placement," *IEEE Transactions on Cloud Computing*, 2020.

[16] B. Shen, X. Xu, L. Qi, X. Zhang, and G. Srivastava, "Dynamic server placement in edge computing toward Internet of vehicles," *Computer Communications*, vol. 178, pp. 114–123, 2021.

[17] D. Kandris, C. Nakas, D. Vomvas, and G. Koulouras, "Applications of wireless sensor networks: an up-to-date survey," *Applied System Innovation*, vol. 3, no. 1, p. 14, 2020.

[18] H.-L. Truong and S. Dustdar, "Principles for engineering IoT cloud systems," *IEEE Cloud Computing*, vol. 2, no. 2, pp. 68–76, 2015.

[19] T. Dinh, Y. Kim, and H. Lee, "A location-based interactive model of Internet of Things and cloud (IoT-Cloud) for mobile cloud computing applications," *Sensors*, vol. 17, no. 3, p. 489, 2017.

[20] C. Stergiou, K. E. Psannis, B. B. Gupta, and Y. Ishibashi, "Security, privacy & efficiency of sustainable cloud computing for big data & IoT," *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 174–184, 2018.

[21] M. Ashouri, P. Davidsson, and R. Spalazzese, "Cloud, edge, or both Towards Decision Support for Designing IoT applications," in *Proceedings of the 2018 Fifth International Conference on Internet of Things: Systems, Management and Security*, pp. 155–162, IEEE, Valencia, Spain, October 2018.

[22] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[23] N. Abbas, Y. Zhang, A. Taherkordi, and Tor Skeie, "Mobile edge computing: a survey[J]," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[24] N. Hassan, S. Gillani, E. Ahmed, I. Yaqoob, and M. Imran, "The role of edge computing in Internet of Things," *IEEE Communications Magazine*, vol. 56, no. 11, pp. 110–115, 2018.

[25] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, "Consolidate IoT edge computing with Lightweight virtualization," *IEEE Network*, vol. 32, no. 1, pp. 102–111, 2018.

[26] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.

[27] S. Yu, R. Langar, X. Fu, L. Wang, and Z. Han, "Computation offloading with data caching enhancement for mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, Article ID 11098, 2018.

[28] A. K. Sangaiah, D. V. Medhane, T. Han, M. S. Hossain, and G. Muhammad, "Enforcing position-based confidentiality with machine learning paradigm through mobile edge computing in real-time industrial informatics," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4189–4196, 2019.

[29] S. Wang, T. Tuor, T. Salonidis et al., "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[30] X. Xu, D. Zhu, X. Yang, S. Wang, L. Qi, and W. Dou, "Concurrent practical byzantine fault tolerance for integration of blockchain and supply chain," *ACM Transactions on Internet Technology*, vol. 21, no. 1, pp. 1–17, 2021.

[31] X. Xia, F. Chen, Q. He et al., "Data, user and power allocations for caching in multi-access edge computing," *IEEE Transactions on Parallel and Distributed Systems*, p. 1, 2021.

[32] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. M. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2651–2664, 2018.

[33] A. A. Al-Habob, O. A. Dobre, A. G. Armada, and S. Muhaidat, "Task scheduling for mobile edge computing using genetic algorithm and conflict graphs," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 8805–8819, 2020.

[34] Y. Zhang, Y. Liu, J. Zhou, J. Sun, and K. Li, "Slow-movement particle swarm optimization algorithms for scheduling security-critical tasks in resource-limited mobile edge computing," *Future Generation Computer Systems*, vol. 112, pp. 148–161, 2020.