You and Tang *Journal of Cloud Computing: Advances, Systems and Applications* (2021) 10:41
https://doi.org/10.1186/s13677-021-00256-4

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things

Qian You and Bing Tang*

Abstract

As a new form of computing based on the core technology of cloud computing and built on edge infrastructure, edge computing can handle computing-intensive and delay-sensitive tasks. In mobile edge computing (MEC) assisted by 5G technology, offloading computing tasks of edge devices to the edge servers in edge network can effectively reduce delay. Designing a reasonable task offloading strategy in a resource-constrained multi-user and multi-MEC system to meet users' needs is a challenge issue. In industrial internet of things (IIoT) environment, considering the rapid increase of industrial edge devices and the heterogenous edge servers, a particle swarm optimization (PSO)-based task offloading strategy is proposed to offload tasks from resource-constrained edge devices to edge servers with energy efficiency and low delay style. A multi-objective optimization problem that considers time delay, energy consumption and task execution cost is proposed. The fitness function of the particle represents the total cost of offloading all tasks to different MEC servers. The offloading strategy based on PSO is compared with the genetic algorithm (GA) and the simulated annealing algorithm (SA) through simulation experiments. The experimental results show that the task offloading strategy based on PSO can reduce the delay of the MEC server, balance the energy consumption of the MEC server, and effectively realize the reasonable resource allocation.

Keywords: Mobile edge computing, Task offloading, Particle swarm optimization, Industrial internet of things

Introduction

The fifth-generation mobile communication system, the so-called 5G, has now become a hot topic in the industry. The main technical challenges that 5G networks need to face are high speed, low end-to-end delay, high reliability, and large-scale connections. In the 5G era, applications such as HD video, virtual reality (VR), augmented reality (AR) will inevitably bring a massive amount of data to the network, which not only brings a heavy load to the backhaul, but also challenges the centralized processing capabilities of the core network. Due to the low latency requirements of 5G, edge computing has also been applied in the industry, whose idea is similar to the concepts of memory and cache in computers. The data frequently used by users are put in the edge of networks closer to users to reduce the delay while reducing the load on the core network.

In order to cope with the problems of insufficient processing capacity and limited resources of smart devices, the industry has introduced the concept of computation offloading in mobile edge computing (MEC) [1, 2]. Computation offloading generally refers to the reasonable allocation of computation-intensive tasks to a proxy server with sufficient computing resources for processing and then fetching the calculated results from the proxy server. Edge computing offloading means that smart devices offload computing tasks to the MEC server. In response to the lack of traditional cloud computing capabilities, edge computing provides cloud computing functions at the edge of the wireless access network near mobile users

*Correspondence: btang@hnust.edu.cn
School of Computer Science and Engineering, Hunan University of Science and Technology, 411201 Xiangtan, China

to meet the needs of rapid interactive response and provide universal and flexible computing services. In order to use the services provided by the edge network, how to offload the tasks undertaken by the smart device to the edge server and make efficient and reasonable offloading decisions is the main research direction of the current edge computing problem.

Industrial Internet of Things (IIoT) is the continuous integration of various acquisition and control sensors or controllers with perception and monitoring capabilities, as well as mobile communication, intelligent analysis, and other technologies into all links of the industrial production process, thereby significantly improving manufacturing efficiency, improving product quality, and reducing products cost and resource consumption. Reliable and fast network transmission and data processing are two important technologies in the IIoT [3–5]. In the IIoT environment, smart devices need to process a large amount of data, and it is not realistic to transmit all of them to the core network for processing. Therefore, how to quickly upgrade traditional industries in the new era of edge computing is a challenge.

In this paper, we mainly consider the 5G IIoT environment, that the 5G smart devices offload a large number of computing tasks to the MEC server in the smart factories. As can be seen, Fig. 1 shows a typical example of 5G IIoT. A computer vision-based product quality detection system is deployed in a smart factory. Cameras are connected to smart devices in order to detect product appearance images. After the model has been trained using machine learning, product appearance images are then detected using the model. By configuring edge servers, detection tasks can be scheduled to edge servers, and data are transmitted through 5G networks. This process is called task offloading. The problems that need to be tackled are how to offload the tasks from smart devices to the MEC servers nearby, and then allocate the MEC server's computing resources to ensure processing efficiency, guaranteeing the task latency requirements. However, latency requirements for task processing are getting higher and higher, so we need to find an optimal computing offloading strategy.

The main contributions of this paper are summarized as follows. We consider the application scenarios of industrial IoT production lines, and we utilize the discrete particle swarm optimization algorithm (PSO) to solve the minimum delay problem in the computation offloading strategy. We use the offloading strategy based on PSO to allocate tasks from smart devices to the MEC server, and finally obtain the task offloading solution with the smallest total cost under the energy consumption constraint. The PSO-based computing offloading strategy is compared with simulated annealing algorithm (SA), and genetic algorithm (GA) to verify the superiority of PSO.

The rest of this paper is organized as follows: The second section surveys related work about edge computing and computation offloading. Then, we present the problem definition, and algorithm implementation, followed by the demonstration of experimental results. The final section concludes the whole paper.

Related work

Mobile edge computing

Mobile edge computing has been emerged as a new paradigm of distributed systems [1, 2]. There are many research topics about MEC, for example, server placement [6], application placement, task offloading [7, 8], network architecture, and so on [9]. In order to use the services provided by the edge network, how mobile devices offload tasks to the MEC server so as to obtain the minimum total task processing cost and obtain a reasonable offloading decision is a popular research direction of MEC. With the development of 5G, it is necessary to combine 5G and artificial intelligence to optimize the management network. A deep data-driven anomaly detection learning framework based on mobile edge computing is presented [10]. However, high energy consumption is troublesome. Ren et al. proposed a novel layered computing architecture to reduce the high energy consumption of mobile augmented reality [11]. Furthermore, A hybrid mobile edge computing platform is considered [12], which aims to minimize the energy consumption of user equipment, and a hybrid online offloading framework based on deep learning is proposed. The best deployment of MEC server and C-RAN for delay-sensitive service coordination, which was studied by Wang et al. [13].

Computation offloading

Aazam et al. proposed the classification of the latest offloading solutions for fog computing, cloud computing, and the Internet of Things [14], which also considered scenarios for performing computation offloading from different factors. In [15], an energy-efficient task offloading method optimized by differential evolution was proposed by Sun et al., which optimizes the energy efficiency of edge computing systems from an energy perspective. A new method of offloading annotations to perform heterogeneous GPU computing in existing workloads with almost no code modification was implemented in [16]. Cha et al. [17] proposed Virtual Edge, a new method to promote collaborative vehicular edge computing. A reverse offload model was developed to reduce the overhead of moving data between different memory areas [18]. In [19], Cheng et al. studied task offloading strategies and wireless resource allocation in multi-user and multi-MEC server systems based on orthogonal frequency division multiplexing access. On the basis of the work in [19], a joint optimization strategy for computing resource

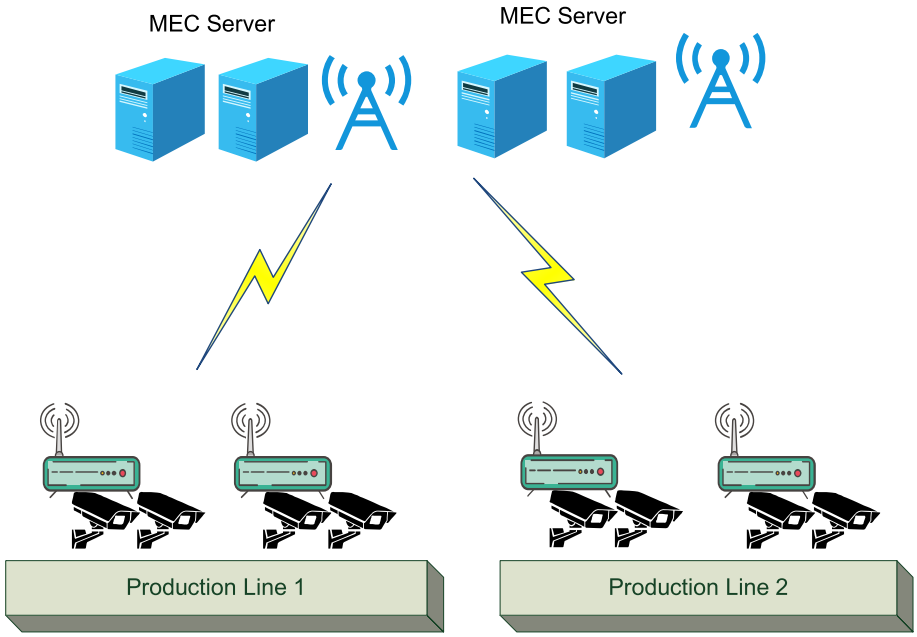

Fig. 1 Edge computing scenarios in 5G Industrial Internet of Things environment

allocation was proposed in [20]. A deep Q-network for multi-agent settings (MADQN) based on the predicted popularity was introduced to solve the caching and offloading problems [21].

Resource allocation

The resource allocation problem of MEC was solved using the optimization framework of reinforcement learning in the context of a multi-user MEC system [22]. A user pairing algorithm based on the maximum difference of classification and sub-maximum from the perspective of downlink non-orthogonal multiple access (NOMA) system allocation capacity gains was presented [23]. In [24], Chen et al. used an asynchronous advantage participant with excellent performance-criticizing learning algorithms to solve complex dynamic resource allocation problems. A new decentralized resource allocation mechanism for vehicle-to-vehicle communication on the basis of deep reinforcement learning was studied [25]. Teng et al. studied the resource allocation problem in ultra-dense network (UDN) [26]. An algorithm based on the alternating direction multiplier method was implemented to obtain the best resource allocation scheme [27]. How to allocate resources to minimize the average service response time was elaborated [28].

In [29], Wang et al. used the PSO and game theory-based MEC task allocation strategies to minimize the time delay. A computing algorithm based on genetic algorithm to allocate computing resources was presented, and the results showed that the algorithm could minimize the energy consumption of user equipment [30]. In [31], Chen et al. studied the data placement strategy of the workflow in MEC, and adopted a method based on genetic algorithm particle swarm optimization, and the final result shows that this method can effectively reduce the data placement cost of MEC.

To sum up, existing research on task offloading pays more attention to whether tasks can be decomposed into subtasks, whether tasks should be offloaded or run locally, how many tasks should be offloaded, and homogeneous edge server scenarios, etc. In the industrial Internet of Things scenario, how to offload tasks generated by smart edge devices to heterogenous edge servers is an challenge issue. However, the existing literature does not model this multi-objective problem well and solve the relationship between offloading cost and time delay well. This is exactly the research work of this paper.

Problem definition

For the Industrial Internet of Things scenarios in the 5G environment, the intelligent production line in the factory environment configures multiple devices such as smartphones, smart cameras, and AR devices, which refers to “multi-user". In the factory environment, due to the data that needs to be processed, whose amount is so large that multiple MEC servers are often needed to handle the tasks of smart devices, which refers to “multi-MEC". This paper is to verify that by using the PSO offloading strategy to find the best position for task offloading, and obtain the

task offloading results, which can minimize the total delay of task processing.

This paper considers the Industrial Internet of Things environment, assuming that there are currently M smart devices and N MEC servers. The tasks on each smart device will be offloaded to a specific MEC server for calculation. We consider only non-divisible tasks, and one smart device submits only one task. For one task, it can be either offloaded to MEC servers or executed locally. Therefore, there are $N+1$ possible positions where each task can be executed. That is, offloading to N MEC servers for execution and local execution. This paper considers three main aspects: the time delay model, energy consumption model, and calculation model. The meanings of symbols used in system modeling are shown in Table 1.

Time delay model

The total delay to complete the i-th task on the j-th MEC server includes the transmission delay and the MEC server calculation delay, which can be calculated as

$$T_i^j = T_{tran,i}^j + T_{mec,i}^j \tag{1}$$

Table 1 Notations

Symbol	Meaning
D_i	The amount of data for the i-th task
C_i	CPU cycles needed to process each bit of data
$C_{u,i}$	CPU frequency of the i-th device
$C_{s,j}$	CPU frequency of the j-th MEC server
W	Transmission bandwidth
S_i	Transmission power of the i-th device
$A_{i,j}$	Channel gain
N_0	Noise power spectral density
$r_{i,j}$	Transmission rate from local device i to the MEC server j
$E_{\max}^j$	Maximum energy consumption constraint of the j-th MEC server
g	Penalty factor
T_i^j	The total delay from local device i to the MEC server j
E_i^j	The energy consumption from the i-th device to the j-th MEC server
$E_{calc,i}^j$	Calculation energy consumption from the i-th device to the j-th MEC server
$E_{tran,i}^j$	Transmission energy consumption from the i-th device to the j-th MEC server
$T_{mec,i}^j$	Calculation delay from the i-th device to the j-th MEC server
$T_{tran,i}^j$	Transmission delay from the i-th device to the j-th MEC server
M	The number of tasks
N	The number of MEC servers

We define that C_i presents how many CPU cycles are needed for each bit of data to be processed, and D_i indicates the amount of data used to process the task. Therefore, the current task amount calculation is known as $D_i * C_i$, and the MEC calculation delay is equal to the task amount divided by the CPU frequency of the MEC server, which is expressed as:

$$T_{mec,i}^j = \frac{D_i * C_i}{C_{s,j}} \tag{2}$$

While considering the transmission delay of MEC, it is necessary to obtain the channel transmission rate from Shannon's formula, and we have

$$r_{i,j} = W * \left(1 + \frac{S_i * A_{i,j}}{W * N_0}\right) \tag{3}$$

where S_i is the transmit power of each local device, $A_{i,j}$ means channel gain from the i-th device to the j-th MEC server, W means transmission bandwidth, and N_0 means noise power spectral density.

Therefore, the transmission delay can be converted into

$$T_{tran,i}^j = \frac{D_i * C_i}{r_{i,j}} \tag{4}$$

Then, the total delay in completing the i-th task on the j-th MEC server is calculated by

$$T_i^j = \frac{D_i * C_i}{C_{s,j}} + \frac{D_i * C_i}{W * \left(1 + \frac{S_i * A_{i,j}}{W * N_0}\right)} \tag{5}$$

Energy consumption model

Energy consumption E_i^j for the i-th task on the j-th MEC server is divided into two parts, calculation energy consumption and transmission energy consumption.

$$E_i^j = E_{calc,i}^j + E_{tran,i}^j \tag{6}$$

The energy consumption of calculation will also be affected by the amount of computing tasks. So, we have

$$E_{calc,i}^j = R_i * U^2 * C_{s,j} * D_i * C_i \tag{7}$$

where R_i depends on the effective switching capacitance, and U means the voltage.

Transmission energy consumption is for the tasks offloaded to the MEC server. Since the transmission delay has been calculated in the delay model, the transmission energy consumption is

$$E_{tran,i}^j = S_i * T_{tran,i}^j \tag{8}$$

where S_i represents the transmit power of each local device.

Then, the total energy consumption is

$$E_i^j = R_i * U^2 * C_{s,j} * D_i * C_i + \frac{D_i * C_i}{r_{i,j}} * S_i \tag{9}$$

Calculation model

The current scenario is to minimize the delay problem. If the queuing delay is not considered, there is a situation: a specific MEC server has better computing performance. Since queuing delay is not considered, most tasks will be offloaded to this MEC server. Hence, a server with better performance will consume much energy, even higher than the maximum energy consumption. In this case, if we offload the task to another MEC server, we can relieve the pressure of offloading the MEC server with better performance. Therefore, we consider adding a penalty function to balance the load of each MEC server.

$$F(X) = \sum_{j=1}^{N}\sum_{i=1}^{M} T_i^j + penalty(X) \tag{10}$$

$$penalty(X) = g * \sum_{j=1}^{N}\sum_{i=1}^{M}\left(E_i^j - E_{\max}^j\right) \tag{11}$$

We define $E_{\max}^j$ as the maximum energy consumption constraint of the j-th MEC server. $X = \{x_1, x_2, ..., x_M\}$ represents the offloading vector, whose size is the same as the number of tasks to be processed. The value of each element x_i is defined as $x_i = 0$ (executed locally) or $x_i \in [1, N]$ (means the id of the MEC server which is used to process the specific task). In order to optimize the algorithm, consider adding weights before the delay and penalty functions. The fitness function is as follows:

$$F(X) = a*\sum_{j=1}^{N}\sum_{i=1}^{M} T_i^j + b*g*\sum_{j=1}^{N}\sum_{i=1}^{M}\left(E_i^j - E_{\max}^j\right) \tag{12}$$

where a and b are two weights that should be customized, and g represents the penalty coefficient. The more energy consumed, the larger the penalty term.

Algorithm implementation

Particle coding

Particle swarm optimization algorithm is an evolutionary computing technology, originated from the study of bird predation behavior. The basic idea of the particle swarm optimization algorithm is to find the optimal solution through collaboration and information sharing between individuals in the group.

The particle swarm algorithm simulates the birds in a flock of birds by designing a massless particle. The particle has only two attributes: speed and position. Speed represents the speed of movement, and position represents the direction of movement. Each particle searches for the optimal solution individually in the search space, and records it as the current individual extreme value, and shares the individual extreme value with other particles in the entire particle swarm, which aims to find the optimal individual extreme value as the current global optimal solution of the entire particle swarm, all particles in the particle swarm adjust their speed and position according to the current individual extreme value they find and the current global optimal solution shared by the entire particle swarm.

Assuming that the task set is *Task*, M represents the number of tasks to be processed, N represents the number of MEC servers to represent the amount of tasks, $E_{\max}^j$ is the energy consumption constraint of the j-th MEC server, $C_s = \{C_{s,1}, C_{s,2}, C_{s,3} \ldots C_{s,N}\}$ represents the CPU clock frequency of the MEC server, and the channel gain matrix is defined as:

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & \ldots & A_{1,N} \\ A_{2,1} & A_{2,2} & \ldots & A_{2,N} \\ \ldots & \ldots & \ldots & \ldots \\ A_{M,1} & A_{M,2} & \ldots & A_{M,N} \end{bmatrix} \tag{13}$$

which is used to calculate the maximum transmission rate when a task is offloaded to of the MEC server. The channel gain $A_{i,i}$ is 0, and $A_{i,j}$ ($1 \le i \le M$, $1 \le j \le N$, $i \ne j$) represents the channel gain required for mobile device i to transmit to the j-th MEC server, $C_u = \{C_{u,1}, C_{u,2}, C_{u,3} \ldots C_{u,M}\}$ represents the CPU frequency of the local device, and $S = \{S_1, S_2, S_3 \ldots S_M\}$ represents the transmit power of each local device.

Particle coding adopts integer coding, and the element of each particle can be any integer between 0 and N. The dimension of the particle is the same as the number of task sets. As we mentioned in previous section, the particle position vector $X = \{x_1, x_2, ..., x_M\}$ is used to indicate that all tasks are offloaded to the corresponding MEC server, whose dimension is the same as the number of tasks to be offloaded and value is initialized randomly. Let's take an example, the current task set has 5 tasks, the task set $Task = \{t_1, t_2, t_3, t_4, t_5\}$. Assuming the particle code is [0, 3, 5, 6, 9], which means, t_1 are directly calculated locally, and t_2 are offloaded to the MEC server whose id is 3 for calculation, and so on.

The particle velocity vector is used to represent the span of offloading tasks to other servers which is denoted by $V = \{v_1, v_2, ..., v_M\}$. First of all, give all particles a random velocity, and update the value of velocity which is rounded before doing the calculation during the iteration process. The dimension of the particle velocity vector is the same as the number of tasks to be offloaded. Let's take an example, the particle position vector X is [5, 3, 1, 8, 6], and the particle velocity vector V is [1, 2, 3, 2, 1]. The first element 5 in the particle position vector X means the task t_1 is offloaded to the MEC server whose id is 5. The first element in the particle velocity vector V is 1, which means in the iterative update process, t_1 is moved to the next MEC server, that is, t_1 is offloaded to the MEC server whose id is 6 for calculation. According to this rule, for

task t_i, it is offloaded to the corresponding server whose id is expressed by $x_i' = (x_i + v_i)\ \%N$.

Fitness function

The fitness function of the particle represents the total cost of offloading all tasks to different MEC servers under the condition of energy consumption constraints, which is expressed by Eq. (12).

Algorithm flow

The algorithm flow chart is shown in Fig. 2, and the detailed algorithm of PSO-based task offloading is demonstrated as follows:

Step 1: Initialization.

Initialize the population, give the particles an initial velocity and an initial position, and set various parameter conditions.

Step 2: Calculate fitness value.

Bring the particle position vector into the fitness function, calculate the particle's fitness value, and save it with *newFitness*.

Step 3: Find the best fitness value of the group.

If *newFitness* is less than the value of fitness, update the fitness value and save the corresponding position vector.

Step 4: Update particle speed and position.

$$x_i = W_p * x_i + c_1 * rand() * (P_{best} - v_i) + c_2 * rand() * (G_{best} - v_i) \tag{14}$$

$$v_i = v_i + x_i \tag{15}$$

The above two equations are used to update the velocity and position of the particles, where W_p is the inertia weight used to adjust the search range of the solution space, P_{best} is the optimal value of the current particle, G_{best} is the optimal value found in the cluster, c_1 and c_2 are the learning factor. In this paper, there are dynamic multi-local calculations and offloading calculation selection problems in the MEC calculation offloading decision. In order to further improve the exploration ability of the PSO algorithm in the solution space, the acceleration factor is improved by the angle, and the static acceleration factor is replaced with a dynamic acceleration factor.

$$c_1 = \frac{\varepsilon}{h} \tag{16}$$

$$c_2 = \eta * h * h \tag{17}$$

where h is the number of iterations, ε represents the individual cognitive impact factor, and the value range in this scenario is [149, 280]; η represents the social cognitive impact factor, and the value range in this scenario is [0.00013, 0.000205].

Step 5: Determine whether the algorithm is over.

When the number of iterations is less than or equal to the maximum number of iterations, repeat Step 2, Step 3, and Step 4; otherwise, the iteration terminates.

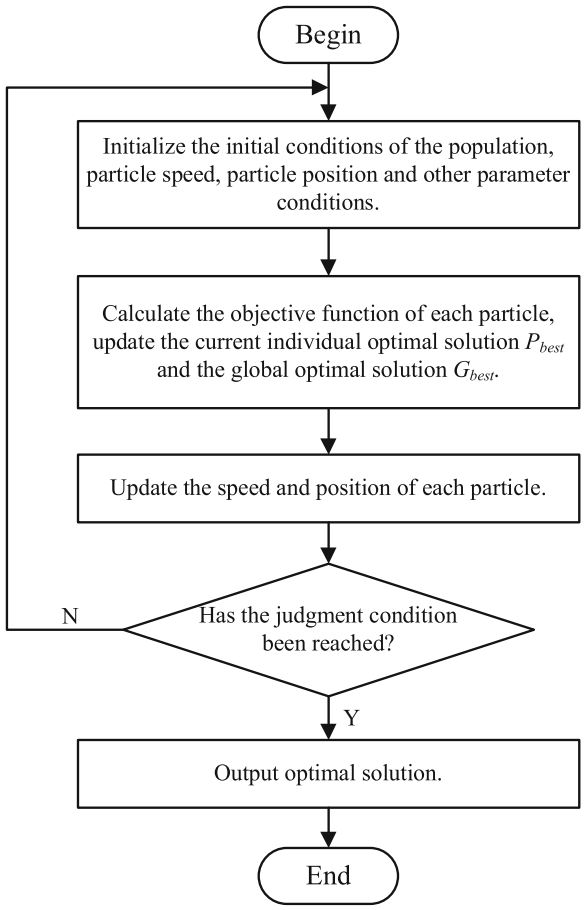

Fig. 2 Algorithm flow chart

Experimental results and analysis

Experimental parameter settings

This section implements the particle swarm optimization algorithm through java language and simulates the task offloading scenario under the Industrial Internet of Things. In this section, the number of MEC servers is 10, and M smart devices respectively initiate offloading requests to 10 MEC servers. The parameter settings are listed in Table 2.

Baseline algorithms

(1) ***Genetic Algorithm***

It is a computational model that simulates the biological evolution process of natural selection and the genetic mechanism of Darwin's biological evolution theory. It is a method of searching for the optimal solution by simulating the natural evolution process. According to the principle of survival of the fittest and survival of the fittest, successive generations evolve to produce better and better approximate solutions after the generation of the first-generation population. In each generation, individuals are

Table 2 Experimental parameter settings

Parameter	Value
D_i	$7-40Mbit$
C_i	$800-1200Cycles/bit$
$C_{u,i}$	$1GHz-3GHz$
$C_{s,j}$	$4GHz-8GHz$
W	$1MHz$
S_i	$100-500mW$
$A_{i,j}$	$2*10^{-6}-2*10^{-5}$
$W*N_0$	$1*10^{-9}W$
$E^j_{\max}$	$1000J$
g	$10^{-2.5}$

selected according to the fitness of individuals in the problem domain, and with the help of natural genetics. The genetic operator combines crossover and mutation to generate a population representing the new solution set. Like the natural population evolution, this process will cause the offspring population more adaptable to the environment than the previous generation. The optimal individual in the last generation population can be decoded as the approximate optimal solution to the problem.

The genetic algorithm includes coding, initializing the population, evaluating the fitness of individuals in the population, selection, crossover, and mutation. The operation of selecting superior individuals in a group and eliminating inferior individuals is called selection. The selection operator is sometimes called the regeneration operator. The purpose of selection is to directly inherit the optimized individual to the next generation or generate a new individual through pairing and crossover and then inheriting it to the next generation. The reorganization and mutation of genetic genes play a central role in the evolution of natural organisms.

Similarly, the critical role in genetic algorithms is the crossover operator of genetic operations. The so-called crossover refers to the operation of replacing and recombining the partial structure of two parent individuals to generate new individuals. Through crossover, the searchability of genetic algorithm can be significantly improved.

(2) ***Simulated Annealing Algorithm***

It comes from the process of crystal cooling. If the solid is not in the lowest energy state, heating and then cooling the solid, as the temperature slowly decreases, the atoms in the solid are arranged in a certain shape to form high-density, low-energy regular crystals, corresponding to the global optimal solution in the algorithm. If the temperature drops too fast, it may cause the atoms to lack enough time to arrange into a crystalline structure, resulting in an amorphous with higher energy, which is the local optimal solution.

The simulated annealing algorithm can be decomposed into three parts: solution space, objective function and initial solution. First, the initial temperature T, the initial solution state S, and the number of iterations L for each T value. In the iterative process, a new solution S' is generated, and then the increment $\Delta t = C(S') - C(S)$ is calculated, $C(S)$ is the cost function. If $\Delta t < 0$ then S' is accepted as the new current solution, otherwise S' is accepted as the new current solution with probability $\exp\left(\frac{-\Delta t}{t}\right)$. If the termination condition is met, the current solution is output as the optimal solution, the program is terminated.

Results and analysis

In this paper, we evaluate three offloading strategies: offloading strategy based on GA, offloading strategy based on SA, and offloading strategy based on PSO. The parameter settings for simulation experiments are described in Table 2. In the simulation experiment, it is assumed that both the local devices and MEC servers can handle tasks. It is also assumed that each task will be executed upon arrival, and there will not be task queuing to increase time delay.

Figure 3 indicates the comparison results of the total system cost of different offloading strategies under different energy consumption constraints. The number of devices M is set to 50, and the number of MEC servers N is set to 10. It can be observed that as energy consumption constraints increase, the total system cost of the three offloading strategies will decrease. When the energy consumption constraint is 0J/ms, SA is much worse than the other two strategies. When the energy consumption constraint is 600J/ms, the cost of SA begins to be lower than that of GA, which is due to the increase in energy consumption constraints and the reduced sensitivity of the model to energy consumption constraints. Therefore, it is

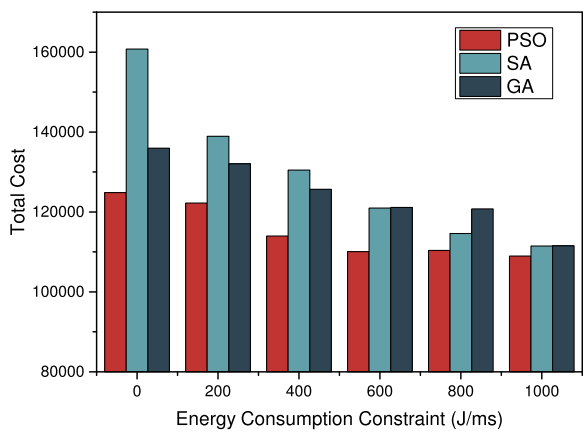

Fig. 3 Comparison of the total cost of the three offloading strategies with different energy consumption constraints

necessary to select appropriate energy consumption constraints. PSO is better than GA and SA under all different energy consumption constraints. When the energy consumption constraint is close to 1000J/ms, the effect of GA, SA and PSO are not much different. Compared with the original system, PSO reduces the total cost by around 12%. When the energy consumption constraint is 0J/ms, the total cost of PSO is 22.3% lower than the total cost of SA and 8.9% lower than the total cost of GA.

We now evaluate the convergence of the three offloading strategies, and the results are shown in Fig. 4. The same as before, we have $M = 50$ and $N = 10$, and the data size of each task is configured as $D_i = 3Mbits$. By changing the number of iterations, we can observe the total delay of all tasks, considering penalty factors.

It can be observed from Fig. 4 that PSO converges faster in the first 10 iterations, and the total system cost remains unchanged after 25 iterations which means the global optimal solution is found. PSO has a strong global optimization capability, which is constantly searching for the global optimal solution in the early stage of the algorithm, and has an excellent global search capability in the later stage. It can also be seen from the figure that the convergence speed of GA and SA is slower than that of PSO. GA converges faster in the first 40 iterations. After 75 iterations, the system finds the global optimal solution, which is close to the PSO algorithm. In contrast, SA algorithm is inferior, which is not completely guaranteed to find the global optimal solution.

The maximal number of devices can be configured. In our experiment, we set the value of maximal number of devices to 250. The comparison results of the total system cost of the three offload strategies with different numbers of the device are shown in Fig. 5. The number of MEC servers N is set to 10, and the data size of each task is $D_i = 3Mbits$. It can be observed that when the number of devices is set to 50, 100, 150, 200, and 250, the average

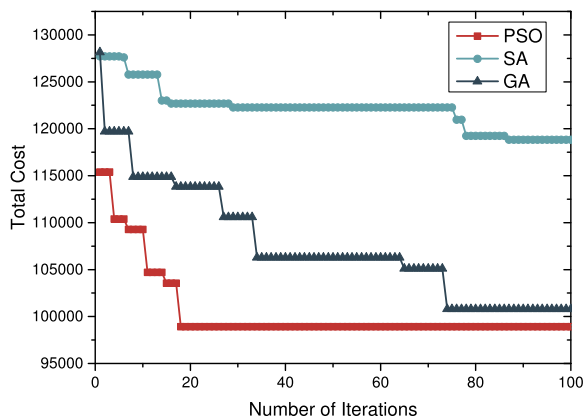

Fig. 4 Comparison of convergence of the three offloading strategies

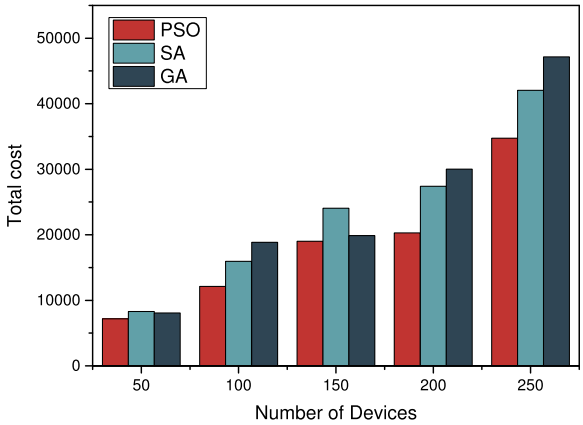

Fig. 5 Comparison of the total cost of the three offloading strategies under different numbers of devices

total delay is increasing, which is due to the increase in the number of devices, the required processing time and transmission time also increase. However, as the number of tasks increases, the increase in the total delay of PSO is less than the increase in the total delay of GA and SA. When the number of devices is less than 150, the results of GA and SA are not much different from that of PSO. When the number of devices is larger than 150, the total system cost of GA began to rise rapidly and is much higher than PSO. It can be observed from the figure that when the number of devices is 250, PSO is superior to SA and GA. The total cost of SA is 17.4% higher than that of PSO, while the total cost of PSO is 26.4% lower than that of GA.

Figure 6 shows the comparison results of the average delay of the three offloading strategies with different energy consumption constraints. When the energy consumption constraint is 0J/ms, the average delay of PSO is much lower than the average delay of GA and SA. As the energy consumption constraint becomes larger, the average delay of the three offloading strategies will decrease. The average delay of PSO has been much lower than GA and SA. When the energy consumption constraint is about 850J/ms, the average delay of SA starts to be lower than that of GA. When the energy consumption constraint is 0J/ms, considering the average delay, PSO is about 10% better than GA and SA.

The convergence of the three offloading strategies based on the average delay has been demonstrated in Fig. 7. When the number of iterations is 5, the convergence speed of GA is faster. However, the global optimal solution was not found. PSO found the global optimal solution after 20 iterations, while the GA and SA strategies stabilized after nearly 50 iterations. In the iterative process, the average delay of PSO is lower than that of SA and GA, which shows the advantages of PSO.

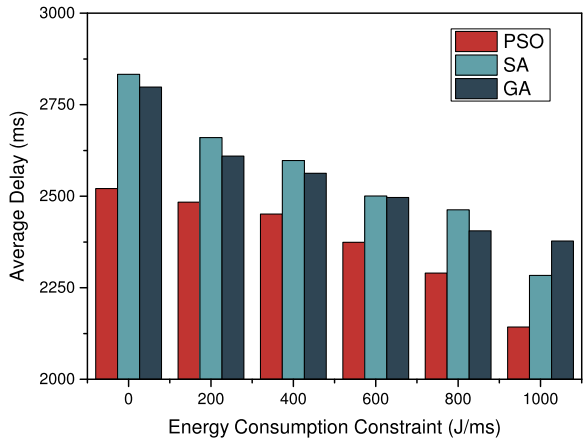

Fig. 6 Comparison of average delay of the three offloading strategies under different energy consumption constraints

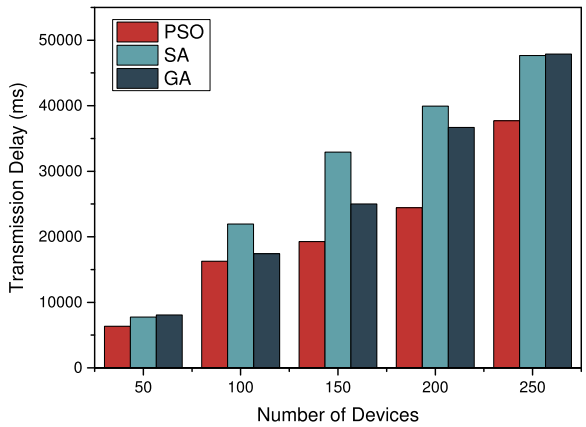

Fig. 8 Comparison of the delays of the three offloading strategies under different numbers of devices

Figure 8 presents the comparison results of the three offloading strategies in terms of transmission delay when the number of devices is varied. The number of devices is increased from 50, 100, 150, 200 to 250. It can be observed from Fig. 8, when the number of devices is between 50 and 100, the execution delays of the three offloading strategies are almost equivalent. With the increase of the number of devices, the superiority of the PSO algorithm appears. As the number of devices increases and the data transmitted increases, the delay will inevitably increase. However, when the number of devices exceeds 150, the execution delay of PSO is much smaller than that of GA and SA. When the number of devices is 200, there are significant differences in the transmission delay between PSO and SA. SA is up to 40% worse than PSO, while the transmission delay of GA is 33.4% higher than that of PSO.

The delay is divided into execution delay and transmission delay. Figure 9 shows the comparison of the execution delay of the three offloading strategies when the number of devices is different. When the number of devices is 50, the execution delay of PSO and GA shows no apparent differences. As the number of devices increases, the execution delay also increases, because the number of devices increases and the amount of data to be processed increases. The extension will become longer. When the number of devices is less than 200, the execution delays of the three offloading strategies are similar. However, the execution delay of PSO is slightly lower than that of the other two algorithms. When the number of devices is larger than 200, the execution delay of GA starts to increase sharply, which is much higher than the execution delay of SA and PSO. Considering the execution delay, when the number of devices is 100, PSO is 16.8% higher than GA and 30.8% faster than SA.

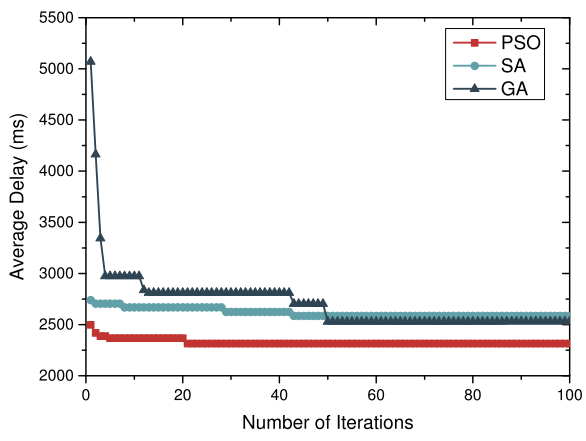

Fig. 7 Comparison of convergence of the three offloading strategies based on average delay

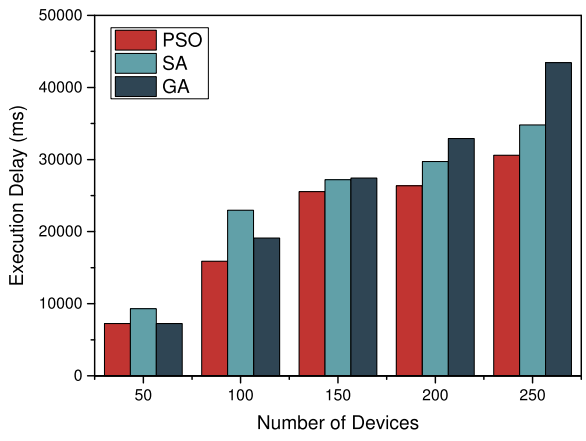

Fig. 9 Comparison of the execution delay of the three offloading strategies under different number of devices

Conclusion

In order to cope with the problems of insufficient processing capabilities of mobile devices and limited resources, the industry has introduced computing offloading into mobile edge computing to solve resource storage and performance problems. In this paper, we introduced three algorithms: particle swarm optimization algorithm, genetic algorithm, and simulated annealing algorithm, and modeled the task offloading problem in the IIoT environment as a multi-user and multi-MEC problem. In order to eliminate the queuing delay of task processing, we added a penalty function to balance the energy consumption and delay. We compare the offloading strategy based on the PSO with the offloading strategy based on the GA and the offloading strategy based on SA. Based on the analysis of experimental results, with the increase in the number of equipment and the number of tasks, the strategy of PSO outperforms the strategy of GA and SA, which can meet the needs of low latency in task processing in the context of industrial Internet of Things. However, it is difficult to control the parameters of PSO. To deal with different problems, how to choose suitable parameters to achieve the best results requires certain experience. This paper simply considers the requirements for low latency and low energy consumption, and does not take the service requirements for high reliability into account.

Abbreviations

MEC: Mobile edge computing; PSO: Particle swarm optimization algorithm; SA: Simulated annealing algorithm; GA: Genetic algorithm; VR: Virtual reality; AR: Augmented reality; IIoT: Industrial internet of things; MADQN: Deep Q-network for multi-agent settings; NOMA: Non-orthogonal multiple access; UDN: Ultra-dense network

Acknowledgements

The authors would like to thank all anonymous reviewers for their invaluable comments.

Authors' contributions

This paper is completed under the supervision of author BT. QY wrote the paper and is responsible for the experiment. BT is responsible for the technical architecture design. The grammar of the paper was reviewed and modified by BT. Finally, BT gives some modification suggestions. All authors have read and approved the final manuscript.

Funding

This work was supported by the National Natural Science Foundation of China under grant no. 61872138 and 61602169, and the Scientific Research Fund of Hunan Provincial Education Department under grant no. 18A186.

Availability of data and materials

The data used to support the findings of this study are available from the corresponding author upon request.

Competing interests

The authors declare that they have no competing interests.

Received: 29 April 2021 Accepted: 14 July 2021
Published online: 28 July 2021

References

1. Abbas N, Zhang Y, Taherkordi A, Skeie T (2018) Mobile edge computing: A survey. IEEE Internet Things J 5(1):450–465
2. Pham Q, Fang F, Ha VN, Piran MJ, Le M, Le LB, Hwang W, Ding Z (2020) A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art. IEEE Access 8:116974–117017
3. Li F, Huang G, Yang Q, Xie M (2021) Adaptive contention window MAC protocol in a global view for emerging trends networks. IEEE Access 9:18402–18423. https://doi.org/10.1109/ACCESS.2021.3054015
4. Huang C, Huang G, Liu W, Wang R, Xie M (2021) A parallel joint optimized relay selection protocol for wake-up radio enabled wsns. Phys Commun 47:101320. https://doi.org/10.1016/j.phycom.2021.101320
5. Liang W, Xie S, Cai J, Xu J, Hu Y, Xu Y, Qiu M (2021) Deep neural network security collaborative filtering scheme for service recommendation in intelligent cyber-physical systems. IEEE Internet Things J. https://doi.org/10.1109/JIOT.2021.3086845
6. Guo F, Tang B, Zhang J (2021) Mobile edge server placement based on meta-heuristic algorithm. J Intell Fuzzy Syst 40(5):8883–8897
7. Tang L, Tang B, Zhang L, Guo F, He H (2021) Joint optimization of network selection and task offloading for vehicular edge computing. J Cloud Comput 10(1):23
8. Tang L, Tang B, Tang L, Guo F, Zhang J (2020) Reliable mobile edge service offloading based on P2P distributed networks. Symmetry 12(5):821
9. Mao Y, You C, Zhang J, Huang K, Letaief KB (2017) A survey on mobile edge computing: The communication perspective. IEEE Commun Surv Tutorials 19(4):2322–2358
10. Hussain B, Du Q, Zhang S, Imran A, Imran MA (2019) Mobile edge computing-based data-driven deep learning framework for anomaly detection. IEEE Access 7:137656–137667
11. Ren J, He Y, Huang G, Yu G, Cai Y, Zhang Z (2019) An edge-computing based architecture for mobile augmented reality. IEEE Netw 33(4):162–169
12. Jiang F, Wang K, Dong L, Pan C, Xu W, Yang K (2020) Deep-learning-based joint resource scheduling algorithms for hybrid MEC networks. IEEE Internet Things J 7(7):6252–6265
13. Wang X, Ji Y, Zhang J, Bai L, Zhang M (2020) Joint optimization of latency and deployment cost over TDM-PON based mec-enabled cloud radio access networks. IEEE Access 8:681–696
14. Aazam M, Zeadally S, Harras KA (2018) Offloading in fog computing for iot: Review, enabling technologies, and research opportunities. Future Gener Comput Syst 87:278–289
15. Sun Y, Song C, Yu S, Liu Y, Pan H, Zeng P (2021) Energy-efficient task offloading based on differential evolution in edge computing system with energy harvesting. IEEE Access 9:16383–16391
16. Yuan G, Palkar S, Narayanan D, Zaharia M (2020) Offload annotations: Bringing heterogeneous computing to existing libraries and workloads. In: Gavrilovska A, Zadok E (eds). 2020 USENIX Annual Technical Conference, USENIX ATC 2020. USENIX Association. pp 293–306. https://www.usenix.org/conference/atc20/presentation/yuan
17. Cha N, Wu C, Yoshinaga T, Ji Y, Yau KA (2021) Virtual edge: Exploring computation offloading in collaborative vehicular edge computing. IEEE Access 9:37739–37751
18. Chen C, Yang W, Wang F, Zhao D, Liu Y, Deng L, Yang C (2019) Reverse offload programming on heterogeneous systems. IEEE Access 7:10787–10797
19. Cheng K, Teng Y, Sun W, Liu A, Wang X (2018) Energy-efficient joint offloading and wireless resource allocation strategy in multi-mec server systems. In: 2018 IEEE International Conference on Communications, ICC 2018, Kansas City, MO, USA, May 20-24, 2018. IEEE, New York. pp 1–6. https://doi.org/10.1109/ICC.2018.8422877
20. Yang X, Yu X, Huang H, Zhu H (2019) Energy efficiency based joint computation offloading and resource allocation in multi-access MEC systems. IEEE Access 7:117054–117062
21. Li S, Li B, Zhao W (2020) Joint optimization of caching and computation in multi-server NOMA-MEC system via reinforcement learning. IEEE Access 8:112762–112771
22. Li J, Gao H, Lv T, Lu Y (2018) Deep reinforcement learning based computation offloading and resource allocation for MEC. In: 2018 IEEE Wireless Communications and Networking Conference, WCNC 2018,

Barcelona, Spain, April 15-18, 2018. IEEE, New York. pp 1–6. https://doi.org/10.1109/WCNC.2018.8377343
23. Islam SMR, Zeng M, Dobre OA, Kwak KS (2018) Resource allocation for downlink NOMA systems: Key techniques and open issues. IEEE Wirel Commun 25(2):40–47
24. Chen M, Wang T, Ota K, Dong M, Zhao M, Liu A (2020) Intelligent resource allocation management for vehicles network: An A3C learning approach. Comput Commun 151:485–494
25. Ye H, Li GY, Juang BF (2019) Deep reinforcement learning based resource allocation for V2V communications. IEEE Trans Veh Technol 68(4):3163–3173
26. Teng Y, Liu M, Yu FR, Leung VCM, Song M, Zhang Y (2019) Resource allocation for ultra-dense networks: A survey, some research issues and challenges. IEEE Commun Surv Tutorials 21(3):2134–2168
27. Leconte M, Paschos GS, Mertikopoulos P, Kozat UC (2018) A resource allocation framework for network slicing. In: 2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018. IEEE, New York. pp 2177–2185. https://doi.org/10.1109/INFOCOM.2018.8486303
28. Zhao L, Wang J, Liu J, Kato N (2019) Optimal edge resource allocation in iot-based smart cities. IEEE Netw 33(2):30–35
29. Wang X, Zhong X, Li L, Lu R, Zheng Y (2019) PSOGT: PSO and game theoretic based task allocation in mobile edge computing. In: Xiao Z, Yang LT, Balaji P, Li T, Li K, Zomaya AY (eds). 21st IEEE International Conference on High Performance Computing and Communications; 17th IEEE International Conference on Smart City; 5th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2019, Zhangjiajie, China, August 10-12, 2019. IEEE, New York. pp 2293–2298. https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00318
30. Guo F, Zhang H, Ji H, Li X, Leung VCM (2018) Energy efficient computation offloading for multi-access MEC enabled small cell networks. In: 2018 IEEE International Conference on Communications Workshops, ICC Workshops 2018, Kansas City, MO, USA, May 20-24, 2018. IEEE, New York. pp 1–6. https://doi.org/10.1109/ICCW.2018.8403701
31. Chen Z, Hu J, Min G, Chen X (2021) Effective data placement for scientific workflows in mobile edge computing using genetic particle swarm optimization. Concurr Comput Pract Exp 33(8)

Publisher's Note