Digital Communications and Networks 8 (2022) 237–246

Contents lists available at ScienceDirect

Digital Communications and Networks

journal homepage: www.keaipublishing.com/dcan

EA-DFPSO: An intelligent energy-efficient scheduling algorithm for mobile edge networks

Yao Lu [a], Lu Liu [a,*], Jiayan Gu [a], John Panneerselvam [a], Bo Yuan [b]

[a] School of Informatics, University of Leicester, University Road, Leicester, LE1 7RH, United Kingdom
[b] School of Computing and Engineering, University of Derby, Kedleston Road, Derby, DE22 1GB, United Kingdom

ARTICLE INFO

Keywords:
Mobile edge computing
Energy-aware systems
Task scheduling
Heuristic algorithms

ABSTRACT

Cloud data centers have become overwhelmed with data-intensive applications due to the limited computational capabilities of mobile terminals. Mobile edge computing is emerging as a potential paradigm to host application execution at the edge of networks to reduce transmission delays. Compute nodes are usually distributed in edge environments, enabling crucially efficient task scheduling among those nodes to achieve reduced processing time. Moreover, it is imperative to conserve edge server energy, enhancing their lifetimes. To this end, this paper proposes a novel task scheduling algorithm named Energy-aware Double-fitness Particle Swarm Optimization (EA-DFPSO) that is based on an improved particle swarm optimization algorithm for achieving energy efficiency in an edge computing environment along with minimal task execution time. The proposed EA-DFPSO algorithm applies a dual fitness function to search for an optimal tasks-scheduling scheme for saving edge server energy while maintaining service quality for tasks. Extensive experimentation demonstrates that our proposed EA-DFPSO algorithm outperforms the existing traditional scheduling algorithms to achieve reduced task completion time and conserve energy in an edge computing environment.

1. Introduction

The recent developments of the Internet of Things (IoT) devices and mobile terminals, and the evolution of Fifth Generation (5G) communication technology, have led to an unprecedented volume of data generation. According to Cisco, IoT connections are anticipated to represent more than half of globally connected devices by 2022 [1,2]. Smart devices (e.g., mobile phones, tablets, desktop computers, smart watches) are currently capable of generating and dealing with massive volumes of computing tasks, as dealing with and managing big data has become the norm [3]. Presently, hand-held mobile devices are efficiently utilized to facilitate and deal with the technology-assisted requirements of daily human life. Although mobile devices (e.g., mobile phones, tablets, smart bracelets) have grown tremendously in recent years, their limitations in battery capacity and computational capability restrain their ability to host heavy-weight computing applications that involve large proportions of data processing, such as real-time video processing and in-memory data processing, etc. [4]. Moreover, delay and latency-sensitive applications require immediate data processing, whereby any undesirable delays incurred in such applications can easily affect the user experience. In this context, always sending the data processing to the back-end servers may not be an ideal choice since communication and processing delays are inevitable [1]. Therefore, from a user experience perspective, orchestrating compute capabilities of the mobile devices and back-end servers can help intense data processing without incurring undesirable delays.

Cloud computing is increasingly used by corporate and individual clients to host intense data processing applications [5,6]. Due to the easy accessibility, low-cost, and efficient computational features of cloud computing services, the number of individual and small-to- large-scale industrial users has increased over the past few years at an unprecedented rate [7]. It has been reported [8] that 92% of global computational workloads are processed in cloud data centers. That is, these increased workloads have caused a series of bottlenecks and network burdens. It is worth noting that the increasing cloud data center deployments leave larger carbon footprints while consuming excessive energy. The network congestion caused by transmitting large amounts of data from terminal devices to cloud data centers necessitates the development of strategic approaches to alleviate this problem.

Edge computing is a recently emerged computing paradigm that

* Corresponding author.
E-mail addresses: yl604@le.ac.uk (Y. Lu), l.liu@le.ac.uk (L. Liu), jg491@le.ac.uk (J. Gu), j.panneerselvam@le.ac.uk (J. Panneerselvam), B.Yuan@derby.ac.uk (B. Yuan).

https://doi.org/10.1016/j.dcan.2021.09.011
Received 25 December 2020; Received in revised form 18 September 2021; Accepted 25 September 2021
Available online 29 September 2021

extends the back-end compute nodes to the network edges [9]. Compared to the traditional cloud architecture, edge computing places the edge nodes in closer physical proximity to the terminal devices, and such nodes can be utilized to greatly reduce the time delay incurred by network transmission [10]. There are many possible implementation schemes for edge computing, one widely recognised scheme is to integrate edge computing with wireless communication Base Stations (BSs) to connect BSs with cloud data centers [11]. Through this design, cloud and edge computing capabilities can be comprehensively utilized for jointly hosting task execution applications without any undesirable delays. Herein, lightweight or delay-sensitive tasks can be transmitted directly from terminals to edge nodes for processing, allowing results to be fed back to users quickly [12]. This strategy can significantly reduce overall processing time since there is no need to transmit tasks to cloud data centers over long distances. Furthermore, the probability of network-related attacks on sensitive data can be reduced. Cloud data centers are used only when the processing requirements of intense heavyweight tasks exceed edge node capacities. Put simply, edge computing cannot be regarded as a substitute for cloud computing, but can be used in conjunction with the cloud architecture in order to enhance the functional efficiency of entire networks, as shown in Fig. 1.

However, many edge devices characterize computational capability and power supply limitations. For the edge nodes represented in Fig. 1, excessive energy consumption not only increases operational costs but also shortens the equipment's service lifetime. Therefore, addressing edge network energy characteristics is imperative. Existing efforts to enhance edge node energy utilization [13–15] have focused on either designing task offloading schemes between cloud and edge nodes or on developing supportive hardware equipment for energy-efficient edge computing. The design of workload scheduling and resource allocation schemes for coordinating edge nodes have not gained enough attention, which can significantly contribute to edge node energy efficiency. To this end, this paper proposes a novel task scheduling algorithm, named Energy-Aware Double-Fitness Particle Swarm Optimization (EA-DFPSO), which is based on an improved Particle Swarm Optimization (PSO) algorithm for achieving energy efficiency in an edge computing environment. Our proposed scheduling algorithm exploits an optimized inertia weight to help EA-DFPSO whilst achieving the global best solution. Important contributions of this paper include the following:

1) Dual fitness functions for energy-efficient scheduling are designed and implanted into the PSO algorithm, with careful consideration paid to task completion time and energy consumption of the edge server nodes.

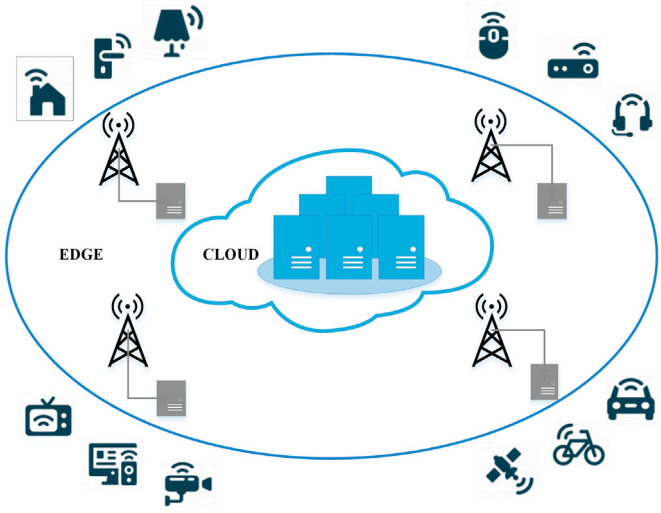

Fig. 1. A three-layer Mobile Edge Computing (MEC) architecture.

2) An optimized inertia weight is used in our proposed novel scheduling algorithm, which helps our proposed heuristic algorithm to achieve the global best solution.
3) An energy-aware scheduling algorithm named EA-DFPSO is developed for edge tasks to reduce edge node energy consumption while ensuring successful task completion.

The rest of the paper is organized as follows. Section 2 reviews the related work. The model description is presented in Section 3, including the energy consumption model and the proposed EA-DFPSO scheduling algorithm. Section 4 introduces the experimental parameter setting and benchmarks. In Section 5, the performance of our proposed scheduling algorithm is evaluated against a few chosen classical scheduling algorithms. Section 6 concludes the paper and outlines our future research directions.

2. Related works

Offloading applications from user devices to edge devices has remained a popular research focus for years. The Refs. [16,17] have proposed efficient distributed algorithms to find out an optimal offloading solution for reducing the time delay of mobile user devices. However, such offloading work can place considerable pressure to edge devices. Thus, exploring efficient methods for edge devices to reduce energy consumption remains an essential research topic.

Several recent studies have investigated energy-efficient schemes for edge computing from various perspectives. The computational limits of the edge devices prevent instructions from being accelerated directly by increasing processor chip frequencies. Special hardware supplements should be implanted into edge nodes for them to process massive volumes of data. To address this issue, a workload partitioning strategy [18] has been proposed for improving the parallel computing abilities of Field-programmable Gate Array–Central Processing Unit (CPU) heterogeneous chips. This strategy uses a scheduler to initially detect the highest throughput on each device for a given application with negligible overhead; the scheduler then partitions the dataset to achieve improved performance.

A user satisfaction-aware energy management model with Dynamic Voltage Frequency Support (DVFS) technology [19] has been proposed, collecting user feedback in the last cycle to improve user satisfaction in the current instance. The subsequent CPU operation sees a recommended frequency provided to adjust the frequency dynamically adjust the frequency after receiving feedback. Another study [20] proposed a Double Deep Q-learning model with DVFS for Energy-efficient Edge Scheduling (DDQ-EES). The proposed model includes a generated network for producing the Q-value for DVFS algorithms and a target network for producing the target Q-values to train the parameters. However, the obvious drawback is that this model will cost much time at the stage of model training.

A Maximum Cache Value (MCV) policy [21] was proposed to manage the cache in edge computing devices. The proposed MVC policy incorporated a model for maximizing the cache hit ratio of mixed memory has been incorporated, along with a hybrid cache management strategy for phase change memory/dynamic random access memory hybrid memory. With this improved two-tier design, the MVC model has efficiently reduced memory access and achieved a higher cache hit rate.

A routing algorithm for energy-aware data transmission [22] was proposed that classified data initially according to properties and priorities, ensuring that the most important data could be transmitted to the appropriate computing node in priority. Data of relatively low importance, such as entertainment data, can also be transmitted effectively under certain conditions (e.g., sufficient bandwidth). This routing algorithm can ensure the data is transmitted to the best node while satisfying the requirements of real-time and reliability.

Another study [23] formulated a joint optimization problem for Quality of Experience (QoE) and energy with fairness. It proposed a

Fairness Cooperation Algorithm (FAC) to obtain an optimal solution for processing data forwarding so as to reduce energy consumption in mobile edge networks.

A framework for task assignment [24] was proposed based on game theory to overcome the delay sensitivity problem of social perception applications. It includes a dynamic feedback incentive mechanism, decentralized virtual gameplay and new negotiation schemes, as well as well-designed private payment functions. Through this framework, the delay sensitivity problem was resolved effectively in social perception applications.

To reduce the energy consumption of edge computing, a system model of Virtual Machine (VM) migration and energy scheduling [25] was proposed, which comprehensively considered task allocation, service migration and energy scheduling to reduce overall energy consumption. To solve this goal optimization problem, the authors designed an algorithm based on low complexity relaxation. This algorithm achieved efficient scheduling of services and resources according to dynamic task requirements and energy consumption.

To achieve the goal of minimizing brown energy consumption, a study [8] investigated VM migration, task allocation, and energy scheduling issues while considering the diversity of green energy supply and user needs. A discrete scheduling optimization model was proposed that proved the Nondeterministic Polynomially (NP) nature of this problem difficulty. In addition, a relaxation-based heuristic algorithm was further developed to solve the unpredictable green energy and user demand dynamic problems [1].

An online and polynomial time complexity model [26], namely Energy Efficient Dynamic Offloading Algorithm (EEDOA), was proposed to solve the flooding issues in MEC to minimize energy consumption, and overcome the latency performance issues of IoT devices. In the absence of prior statistical knowledge related to task arrival or channel condition, this EEDOA model achieved an arbitrary trade-off between transmission energy efficiency and queue backlogs through their experiments. This ensured an upper limit to queue backlogs whilst achieving near-optimal transmission energy consumption. However, due to a lack of support from prior statistical knowledge, it is hard to get optimal solutions for energy saving at the beginning of experiments.

3. Math formulas

This section describes our proposed PSO-based metaheuristic task scheduling algorithm for achieving energy efficiency in mobile edge computing environments. The PSO algorithm is utilized due to its low computational complexity; it requires no crossover or mutation operations, contrasting with other bio-inspired algorithms, such as the ant colony optimization and genetic algorithm. Low computational complexity also reflects low time cost in the edge nodes, which is vital for resource-constrained edge computing environments.

3.1. PSO algorithm math model

Suppose that the size of a given particle is n and the search space is in d dimension. The position vector of the i^{th} particle at the time t is

$$X_i(t) = (x_{i1}, x_{i2}, L, x_{id}), i \in [1, n] \tag{1}$$

The optimal position of the i^{th} particle experience is

$$P_i = (p_{i1}, p_{i2}, \ldots, p_{id}),\ i \in [1, d] \tag{2}$$

The local optimal position is

$$P_j = (p_{j1}, p_{j2}, \ldots, p_{jd}) \tag{3}$$

The velocity vector of the i^{th} particle is

$$V_i = (v_{i1}, v_{i2}, \ldots, v_{id}) \tag{4}$$

The update policy for the particle's velocity and position is given below:

$$V_i(t+1) = \omega \times V_i(t) + D_1 \times R_1 \times [P_i - X_i(t)] + D_2 \times R_2 \times [P_j - X_j(t)] \tag{5}$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{6}$$

where D_1 and D_2 are the learning factors, R_1 and R_2 are random numbers between 0 and 1, respectively, t represents an iteration, and ω is the inertia weight (which is set to a value of 1 by the standard PSO).

In the speed formula (5), the second part $D_1 \times R_1 \times [P_i - X_i(t)]$ is the particle's self-cognition, which is usually the distance between the particle's current position and the best position it has experienced previously. Therefore, if the value D_1 becomes larger, the particle thinks more about itself, itself, improving its own local search ability. The third part of (5), $D_2 \times R_2 \times [P_j - X_j(t)]$, gives the distance from the particle to the social learning information, usually the current position of the particle and the best position obtained by its belonging group, according to the experience of the population, depicting its information-sharing ability and the degree of cooperation among the particles. Therefore, if D_2 becomes large, the particle learns more about its group, which can enhance the global search ability of the particles. Since PSO is a good optimization algorithm, it can help adjust the global and local search capabilities of the particles by dynamically changing the learning factors.

Usually, the position changing range of a given particle is set to $[X_{\min,d}, X_{\max,d}]$ and the speed changing range is limited to $[-V_{\max,d}, V_{\max,d}]$. If the position and speed exceed the boundary value during an iteration, the position and speed of the corresponding particle are updated as the new maximum boundary value for future iterations.

3.2. An improved PSO with inertia weights

In the traditional PSO algorithm [27], the inertia weight ω is usually set as a fixed constant. An unsuitable inertia weight value can seriously affect the efficiency and readiness of the PSO algorithm; if the value is too large, the algorithm may miss the optimal solution, while a ω is too small to affect the update operation of the particles. To address this problem, we propose optimizing ω with the following:

$$\overline{X}(t) = [\overline{x}_1(t), \overline{x}_2(t), \cdots, \overline{x}_n(t)] \tag{7}$$

$$\overline{x}_j(t) = \frac{1}{n}\sum_{i=1}^{n} x_{ij}(t) \tag{8}$$

$$DIS(t) = \frac{1}{nL}\sum_{i=1}^{n}\sqrt{\sum_{j=1}^{d}\left[x_{ij}(t) - \overline{x}_j(t)\right]^2} \tag{9}$$

$$\omega(t) = e^{-DIS(t)}\left(1 - \frac{t}{T_{\max}}\right) + \mu \tag{10}$$

where L is the largest possible distance between particles in the solution domain, the average center of the particle swarm is $\overline{X}(t)$, the distance between the particle and the center is $DIS(t)$, t is the current iteration, T_{max} is the maximum iteration, and μ is a constant value that is used to prevent speed from reaching a value of zero.

The average distance shows the distribution of particles, which can be recognised as a representation of diversity. When the particle swarm diversity is lost rapidly (especially during the early stage of the algorithm implementation), the inertia weight w should be increased in time to improve the particle speed, thereby increasing the population search range, ensuring population diversity, and preventing the algorithm from falling into the local optimal solution. By contrast, when the particle swarm diversity keeps increasing, w should be appropriately reduced, thereby enhancing the local search ability of the algorithm and accelerating its convergence.

3.3. Workload scheduling in MEC

The functional mechanism of edge computing is similar to the traditional cloud computing in various senses [28]. Task scheduling in a cloud environment often uses distributed parallel processing technology, such that cloud jobs submitted by users are divided into several tasks. Through the use of a scheduling algorithm, each task is assigned to a virtual computing node mapped through virtualisation technology and executed in different computing nodes. When all tasks are successfully executed, the results of all tasks belonging to a given job are accumulated and sent to the customer. This typical map-reduce concept is also applicable to an edge computing environment. Its mathematical model for task scheduling is described as follows: users submit their jobs of various needs to the edge nodes. Each job is then divided into a respective number of independent tasks. All tasks are assigned to different virtual resource nodes by task schedulers in the edge environment for execution according to a certain scheduling algorithm, though containerization technology is popular in edge networks. Fig. 2 illustrates a system diagram of the detailed scenario of task scheduling in MEC.

The disadvantages of containers remain obvious compared to VMs. For tasks that require high security or performance, container technology may not be the best choice because tasks in different containers cannot be isolated strictly. We believe that, for the foreseeable future, containers and VMs may coexist in edge networks. Due to the superiority of the VM technology in some aspects, like isolation security and application independence, we assume that nodes mentioned here are VMs. Suppose that a job J encompasses Ta number of tasks that require C resources in the VMs for successful execution. Further, assume that the total number of jobs is u, the total number of tasks is n, and the total number of resources is m. This job can be expressed as a composite $J = \{j_1, j_2, \cdots, j_i, \cdots, j_u\}$, where j_i is the i^{th} job submitted by a user to an edge node and t_{ij} is thej^{th} ask in the i^{th} job. The tasks among the total number of jobs can be expressed as a composite $Ta = \{t_1, t_2, \cdots t_n\}$, and the set of resources can be expressed as a composite $C = \{c_1, c_2, \cdots c_m\}$. In general, the number of available resources is usually less than the number of tasks (i.e., $m < n$). Task scheduling is the process of allocating n tasks among m computing nodes for execution. The ultimate goal of our study is to find the optimal task allocation scheme, so the execution duration and the energy consumption are maintained at the minimum possible level.

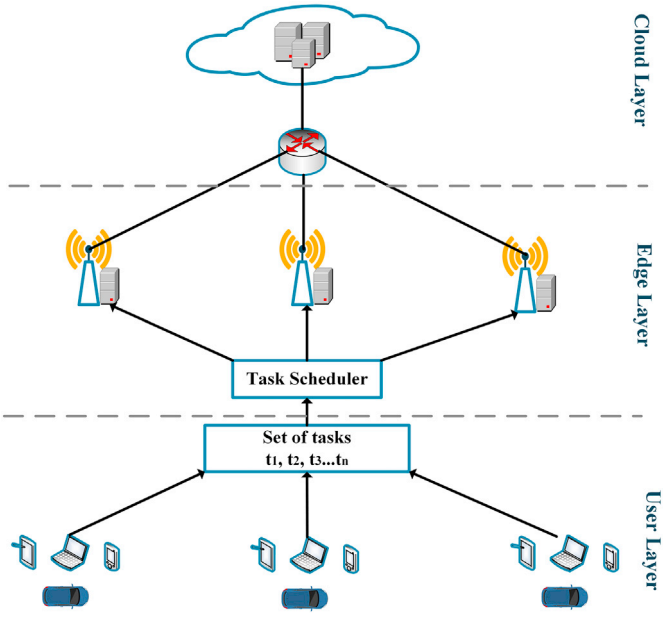

Fig. 2. System diagram of task scheduling in MEC.

3.4. Problem encoding

In the PSO algorithm, the particle encoding methods can usually be divided into direct encoding and indirect encoding. Our proposed improved PSO algorithm in this paper encodes particles using indirect encoding. In this solution, the resources occupied by each task are encoded. The number of tasks to be processed determines the final length of encoding, encoding, where single tasks correspond to single resource nodes and single particles correspond to a task allocation scheme.

The following section explains the encoding process of particles in detail with an example. Suppose that the number of jobs is 3, and the number of tasks contained in job1, job2, and job3 is 3, 3, and 4, respectively (i.e., there are 10 tasks in total). Suppose further that there are 5 differentiated virtual compute nodes in the edge environment. The task scheduling algorithm should reasonably place 10 tasks among 5 VMs for execution. Since each particle can represent a scheduling scheme, particle dimensions are determined by the number of tasks. In this example, each particle can be considered as a ten-dimensional vector. Assuming that one of the particles is represented by Pa (4, 5, 3, 1, 4, 2, 5, 1, 5, 5, 2), feasible encoding and decoding schemes are shown in Tables 1 and 2. It should be noted that tasks are renumbered according to the order of job submission. The encoding process is represented in equation (11), where the value of $Task(i,j)$ represents the total number of tasks and $JobNum(k)$ indicates the count of tasks included in thek^{th}job. In Equation (12), TaskNum expresses the total number of tasks contained across all the submitted jobs and $JobNum(i)$ indicates the number of tasks included in the i^{th} job. For example, Task (2, 1) = 4 in Table 1, which indicates that the Task ID of the first task contained in job 2 is 4. Table 1 indicates this task will be allocated to VM 1 running in the edge node. Table 2 presents the mapping of tasks onto VMs and illustrates the total number of tasks being executed on a given VM.

$$Task(i,j) = \sum_{k=1}^{i-1} JobNum(k) + j \tag{11}$$

$$TaskNum = \sum_{i=1}^{n} JobNum(i) \tag{12}$$

The purpose of task scheduling is to optimize resource utilization. To achieve this goal, we must first understand the implementation of each task, including the completion time and the amount of energy consumption.

Let T_{ij} be the execution time of i^{th} task in VM c_j. The value of T_{ij} depends on the length of the task t_i and the computational capabilities of VM c_j, which is shown in Equation (13). Therefore, the execution time of all tasks on a given VM can be represented by a matrix $T_{n\times m}$, which is shown in Equation (14).

$$T_{ij} = \frac{length(t_i)}{speed(c_j)} \tag{13}$$

Table 1
An example of particle encoding.

Job ID	Task ID	VM ID
1	1	4
	2	5
	3	3
2	4	1
	5	4
	6	2
3	7	5
	8	1
	9	5
	10	2

Table 2
An example of particle decoding.

Task ID	VM ID
4	1
8	
6	2
10	
3	3
1	4
5	
2	5
7	
9	

$$T_{n\times m}=\begin{bmatrix} T_{11} & T_{12} & \cdots & T_{1m} \\ T_{21} & T_{22} & \cdots & T_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ T_{n1} & T_{n2} & \cdots & T_{nm} \end{bmatrix} \tag{14}$$

Suppose that the set of tasks assigned to computing node j is M, the maximum completion time of j can be expressed as

$$T_j=\sum_{i\in M_j} T_{ij} \tag{15}$$

After all the computing nodes (VM) have completed their tasks, the processing time of the node characterising the longest execution time is regarded as the completion time of all jobs, as in the case of parallel tasks [18]. Thus, the completion time of jobs can be represented as follows:

$$T_{Max}=\max(T_j)=\max(\sum_{i\in M_j} T_{ij}) \quad j\in\{1,2,\cdots,m\} \tag{16}$$

Assuming that task t_i consumes energy per unit time in the computation node c_j as p_j, then we can derive the following equations:

$$e_{ij}=T_{ij}\times p_j \tag{17}$$

$$E_j=\sum_{i\in M_j} t_{ij}\times p_j \tag{18}$$

$$E_{total}=\sum_{j=1}^{m}\sum_{i\in M_j} t_{ij}\times p_j \tag{19}$$

where e_{ij} is the energy consumption of task t_i processed in VM c_j, E_j is the total energy consumption of all the tasks processed in VM c_j, E_{total} is the total energy consumption of all the tasks processed in the created VMs of the corresponding edge node.

3.5. Particle swarm initialization

Assume that the size of the population is S, the number of jobs executed in the edge nodes is$u$, the number of tasks divided into n, and the number of virtual computing resources is m. The initialization process can then be described as follows:

The system randomly generates S particles. Each particle can be represented by a multi-dimensional vector. For example, the i^{th} particle can be expressed as

$$X_i=(x_{i1},x_{i2},\cdots,x_{in}),\ 1\le i\le S \tag{20}$$

where x_{ij} indicates the virtual computing resource processing the j^{th} task.

The speed of a particle can also be expressed with a multi-dimensional vector. For example, the speed of the i^{th} particle can be expressed as

$$V_i=(v_{i1},v_{i2},\cdots,v),\ 1\le i\le S \tag{21}$$

The particle's initial positions are random integers belonging to [1, m]. The particle's speed is random integers belonging to [- (m - 1), (m - 1)].

3.6. Energy consumption model

When a task runs on a physical node, the energy consumption mainly encompasses the processor's computational and data transmission power. It is worthy of note that any compute node incurs idle power consumption even when no tasks are running on it. The dynamic power P_D consumed by a compute node while executing a task t_i at frequency f_j and supply voltage V_{dd}^j is computed as follows:

$$P_{D_i}=C_{eff}^i(V_{dd}^j)^2 f_j\sqrt{b^2-4ac} \tag{22}$$

where C_{eff}^i is the switching capacitance while executing the task t_i .

When executing the task t_i, the static power dissipation can be computed as follows:

$$P_{S_i}=L_g(V_{dd}^j K_3 e^{K_4 V_{dd}^j} e^{K_5 V_{bs}}+|V_{bs}|I_{jn}) \tag{23}$$

where K_3, K_4, K_5 are technology specific CPU parameters, L_g is the number of processor logic gates, and V_{bs} and I_{jn} denote the bias voltage and leakage current, respectively [29].

According to the above description, the total power P_i consumed by a compute node while executing a task t_i is the summation of P_D and P_S, which is shown as follows:

$$P_i=C_{eff}^i(V_{dd}^j)^2 f_j+L_g(V_{dd}^j K_3 e^{K_4 V_{dd}^j} e^{K_5 V_{bs}}+|V_{bs}|I_{jn}) \tag{24}$$

The execution time T_{ij} of a task t_i executing at a frequency f_j on the processor pe_k in a computing node c_j is given as follows:

$$T_{ij}=\frac{CC_{ij}}{f_j} \tag{25}$$

where CC_{ij} denotes the clock cycles of a task t_i executing on the processor pe_k. If the value of CC_{ij} cannot be computed directly, T_{ij} can be computed using Equation (13). Therefore, the energy consumption e_{ij} of a task t_i executed on a processor pe_k in a compute node operating at a supply voltage V_{dd}^j is determined by Equations (13) and (24), which are summarized as [29]:

$$e_{ij}=C_{eff}^i(V_{dd}^j)^2 f_j+L_g(V_{dd}^j K_3 e^{K_4 V_{dd}^j} e^{K_5 V_{bs}}+|V_{bs}|I_{jn})T_{ij} \tag{26}$$

In our scheduling scheme, every particle represents a feasible scheduling scheme, making particle quality assessment crucial. In the PSO algorithm, the fitness function is usually used to evaluate the pros and cons of the particles. Each particle has a fitness value to indicate the pros and cons of other particles, indicating the quality of their scheduling scheme.

The fitness function of the PSO algorithm is usually unique, presenting different values for different problems. As such, the design of the fitness function should be carefully considered while solving the scheduling problem. To improve the resource utilization of edge nodes and reduce their energy consumption, this paper proposes a dual fitness function, such that the energy consumption of the edge nodes is reduced in consideration of their completion time. The specific fitness function used is defined as follows:

According to Equations (15) and (16), the first fitness function can be represented as

$$Fit_1(i)=\frac{1}{T_{Max}}=\frac{1}{\max(\sum_{i\in M_j} t_{ij})} \tag{27}$$

$$1\le i\le S;\ j\in\{1,2,\cdots,m\}$$

According to Equations (17)–(19), the second fitness function can be represented as

$$Fit_2(i) = \frac{1}{E_{Total}} = \frac{1}{\sum_{j=1}^{m} \sum_{i \in M_j} T_{ij} \times e_j} \tag{28}$$

The larger the value of fitness function Fit_1, the shorter the total task completion time. Similarly, the larger the value of Fit_2, the smaller the energy consumed by the edge node while completing the jobs. With this dual fitness function, the particle population includes not only the particles characterzing shorter total task completion time, but also the particles characterizing less energy consumption. Through a continuous iteration and selection process, particles performing both on completion time and energy consumption are chosen for task scheduling.

3.7. Particle attribute settings

In particle swarm optimization, each particle has two basic properties: position and velocity. The particle's position indicates the distance relationship between the particle and the target. Speed represents the direction and speed of particle movement. As mentioned in Section 3.5, each particle has an adaptation value determined by the objective function and knows both the best position experienced as well as its current position. This process can be regarded as the particle's memory of its own flight experience. Specifically, the next position update is determined by comparing the fitness value of the current position of a given particle with the fitness value of the optimal position of the previous particle. In addition, each particle is aware of the best position that all particles have experienced in the entire population at a given point in time. This process presents insights into a given particle's experience with its peers. Particles determine their next movement through their own experience and the best experience of their peers, which is achieved by comparing their corresponding fitness value with their peer counter.

Suppose the position of particle i at time t is

$$X_i(t) = (x_{i1}, x_{i2}, \cdots, x_{in}) \tag{29}$$

The best position of particle i experienced in the past is

$$Pbest_i(t) = (pb_{i1}, pb_{i2}, \cdots, pb_{in}) \tag{30}$$

The best position that all particles have experienced in the past is

$$Gbest(t) = (gb_1, pb_2, \cdots, pb_n) \tag{31}$$

The particle update rule is shown in the following equations:

$$Pbest_i(t+1) = \begin{cases} Pbest_i(t), Fit_2(X_i(t+1)) < Fit_2(Pbest_i(t)) \\ Pbest_i(t), Fit_2(X_i(t+1)) = Fit_2(Pbest_i(t)) \cap \\ Fit_1(X_i(t+1)) < Fit_1(Pbest_i(t)) \\ X_i(t+1), Fit_2(X_i(t+1)) = Fit_2(Pbest_i(t)) \cap \\ Fit_1(X_i(t+1)) > Fit_1(Pbest_i(t)) \\ X_i(t+1), Fit_2(X_i(t+1)) > Fit_2(Pbest_i(t)) \end{cases} \tag{32}$$

Let

$$F(t) = \lambda Fit_1(t) + (1-\lambda) Fit_2(t) \tag{33}$$

Objective function:

$$Max: \ F(t) = \lambda Fit_1(t) + (1-\lambda) Fit_2(t) \tag{34}$$

$$Subject\ to \ \sum_{i=1}^{s} x_{ij}, \forall j = 1, \ldots n \quad x_{ij} \in \{0,1\}, \forall i, j$$

$$F_{Max}(t) = Max[F(Pbest_1(t)), F(Pbest_2(t)), \cdots, F(Pbest_S(t))] \tag{35}$$

$$Gbest(t) = F_{Max}(t) \tag{36}$$

where the value of the function $F_{Max}(t)$ is the value of $Pbest_i(t)$ corresponding to the maximum value of the function $F_{Max}(t)$, which is also the global optimal particle position after t number of iterations.

According to Equations (5) and (6), a particle continuously updates its particle speed and position. At the same time, the local optimal position and the global optimal position of the particles are continuously updated. The direction of this update process always advances towards the global optimal solution, and the update continues until either the final global optimal solution is found or the upper limit of the number of iterations is reached. At this time, the solution of the system is regarded as the optimal solution.

3.8. EA-DFPSO algorithm description

To better understand the execution of our proposed EA-DFPSO algorithm in edge environments for energy efficiency, Algorithm 1 describes the particle selection process in detail. The proposed scheduling algorithm uses a fitness function for reducing energy consumption whilst completing all the tasks, as shown in Equations (28) and (29). An improved inertia weight is designed for the PSO algorithm to reach optimal completion time and energy consumption, as shown in Equations (9) and (10).

Algorithm 1. EA-DFPSO

1: Input: number of edge servers, number of tasks, task size, maximum iteration, the parameters C1 and C2.
2: Output: completion time and energy consumption of task executions.
3: Initialisation: The edge environment initialisation with task set {ti} and VMs {Cj}, i ∈ {1, ...m}, j ∈ {1,...n}, parameters of every edge computing node including CPU frequency, RAM, storage and so on. DFPSO algorithm initialisation with speed and position of particles according to Equation (20) and (21)
4: Compute the completion time Tj of every particle
5: Update Pbesti and Gbesti according to the update rule in Equation (33)
6: Update the speed of the particle according to Equation (5) and (10)
7: Update the position of the particle according to Equation (6)
8: Determine if the optimal condition is reached: if yes, stop iteration; else return to Step 5.
9: Output Results

4. Experiments

4.1. Experiments setting

The hardware environment used in the experiments includes the following: CPU - Intel Core i7-8550U @1.8Hz, 8G memory. The simulation processes are mainly conducted on Cloudsim 4.0 [30].

In the simulated edge environment, the computing capability of the nodes (VM) is set to both 100 and 1000 MIPS. Every edge server can have one or more processors, and every task has a random length ranging from 10 to 510 MIPS.

The related parameters of EA-DFPSO are set as follows: the population size is 25, the acceleration constants of D_1 and D_2 are 2, the maximum speed is 1.0, the number of tasks is set to 40, 80, 120, and 160, and the maximum iterations is 500. Other experimental parameter settings will be described in the following sections (if applicable).

4.2. Benchmarks

4.2.1. Short Job First (SJF)

SJF [31] scheduling algorithm prioritizes short jobs first, selecting one or more jobs with the shortest estimated running time from the waiting queue to execute them first. This algorithm is beneficial to the completion of short tasks, but it is very unfriendly to long tasks. If there are a large number of short tasks in the task queue, the completion time of long tasks is delayed significantly. In addition, the algorithm often does not consider task urgency, rendering it unable to handle emergency events flexibly.

4.2.2. First Come First Service (FCFS)

FCFS [32] is one of the simplest task scheduling algorithms. When this algorithm is used for task scheduling, tasks are scheduled based on the order they arrive in the queue, with jobs processing the longest waiting time in the queue being given priority. Regardless of the anticipated execution time, this algorithm selects a few jobs that entered the queue first to begin execution. This scheduling algorithm is beneficial to the completion of long tasks and can ensure that the resources of long tasks are not preempted before they are executed. However, short tasks can experience significant waiting time when large proportions of long tasks occupy the queue. As a result, short tasks cannot be completed in a timely manner, and the total number of tasks completed by this scheduling algorithm within a fixed period of time will also decrease.

4.2.3. Round Robin

Round Robin [33] algorithm sets a fixed time slot for the scheduler by specifying the length of time for each task execution prior to executing the following task. If a given task characterizes less time than the set time slot, then the corresponding task can be completed. If not, only a part of this task is executed, and the remainder of this task is to be held until its next turn. Thus, all the tasks are executed based on this Round Robin fashion. The advantage of this algorithm is its simplicity of stateless scheduling, as it eliminates the need to record the status of all current connections. However, when the request service interval time is relatively large, the scheduling algorithm can easily cause load imbalance between servers [33].

5. Experiment performance evaluation

5.1. Evaluation of execution time for different numbers of tasks

We evaluated the performance of our proposed scheduling algorithm in an edge environment with a large number of experiments. An initial evaluation was carried out to evaluate the execution time for a range of jobs encompassing different numbers of tasks when using our EA-DFPSO scheduling algorithm in the edge environment. At the same time, the performance of our proposed scheduling algorithm was evaluated against three resource scheduling algorithms: SJF, FCFS and Round Robin. Furthermore, the impact of the encompassing number of tasks upon the completion time with a fixed number of VMs was explored.

In this experiment, the total number of VMs in the edge servers was set to 10. It should be noted that the task completion time presented here is only the execution time of the submitted tasks in the edge servers, which does not include the transmission time for sending tasks from the client terminal to the server. This disregards the impacts of network bottlenecks since the scheduling algorithms do not have to control network overheads. The main motivation of this paper is to evaluate the scheduling algorithm performance, placing the transmission delay beyond the scope of this article. All the experiments in this paper were conducted in the same bandwidth communication channel.

Figs. 3 and 4 depict the execution time of different tasks for the four studied scheduling algorithms. Fig. 3 shows an obvious phenomenon: increasing the number of tasks submitted to the edge nodes increases in their completion time for all four scheduling algorithms, which reflects real-life scenarios. At the same time, it can be observed from Fig. 4 that our proposed algorithm EA-DFPSO outperformed the other three scheduling algorithms when executing four sets of different numbers of tasks, exhibiting completion time of 20.91 s, 28.81 s, 32.94 s, and 40.29 s for 40, 80, 120, and 160, respectively. This is because our proposed EA-DFPSO algorithm uses the reciprocal of the task completion time as its fitness function. Herein, our proposed scheduling algorithm fully considers the use of VM resources and server task lengths during the execution process, thereby formulating an ideal scheduling plan. Even if the global optimal solution is not achieved every time, relatively ideal scheduling is obtained in most cases. Figs. 3 and 4 indicate that when the number of tasks is small, the performances of the four scheduling algorithms present no obvious difference. For example, given 40 tasks, the execution time difference between any two algorithms is less than 4.1s, and FCFS exhibits the longest execution time of 24.98s, and our proposed EA-DFPSO exhibits the shortest execution time of 20.91s. However, as the number of tasks increases, the superiority of our proposed EA-DFPSO algorithm is gradually manifested (e.g., when 160 tasks were used, EA-DFPSO saved almost 10 s compared with SJF). This demonstrates the SJF disadvantage of being unable to guarantee timely scheduling for long tasks. In parallel computing, the final completion time of a given job is determined by the long-running tasks.

5.2. Evaluation of task execution time on different numbers of VMs

This section evaluates the performance of our proposed EA-DFPSO scheduling algorithm for a fixed number of tasks with different numbers of VMs against the traditional scheduling algorithm. The total number of tasks was set to a fixed number of 40, the number of VMs was increased from 10 to 30, and each iteration was increased by 5. Figs. 5 and 6 illustrate the time consumed during algorithm execution, indicating significant reductions as the number of VMs increased regardless of the algorithm used. More VM resources in the edge environment ensure more efficient parallel processing, making final job completion times dependent on the number of active VMs. Figs. 5 and 6 clearly indicate that our proposed EA-DFPSO algorithm produced the fastest possible completion time when executing 40 tasks. Given 30 VMs, the EA-DFPSO algorithm could perform all tasks in only 5.64 s, while FCFS required 11.35s, more than twice the former. The other two scheduling algorithms required at least 30% more time to complete the tasks than our proposed algorithm. This is because our algorithm iteratively sought the best scheduling solution according to the fitness function value, which helped reduce the completion time while conserving energy.

5.3. Evaluation of energy consumption

This section evaluates the energy performance of our proposed EA-

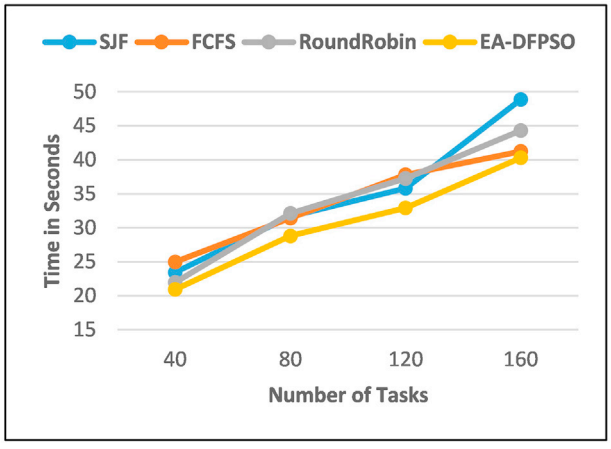

Fig. 3. Execution time for different numbers of tasks.

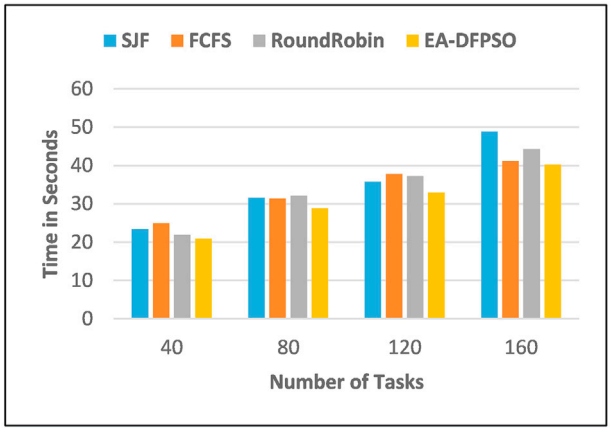

Fig. 4. Execution time for different numbers of tasks.

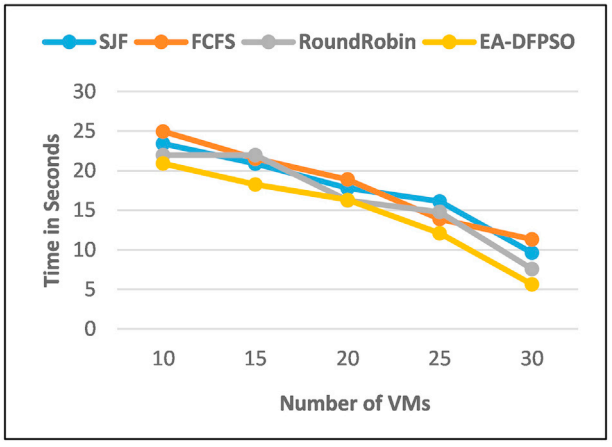

Fig. 5. Impact of the number of VMs on tasks completion time (tasks = 40).

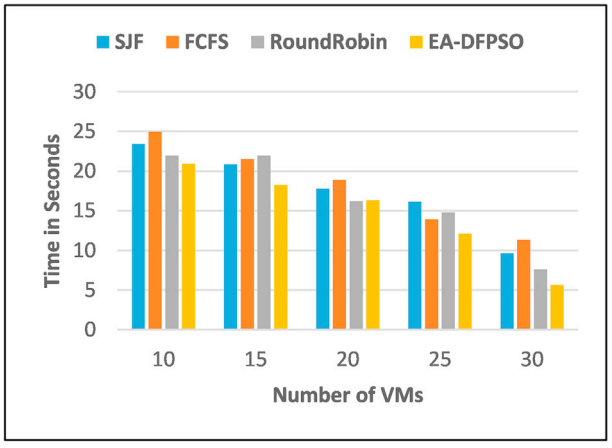

Fig. 6. Impact of the number of VMs on tasks completion time (tasks = 40).

DFPSO algorithm in the edge environments against the other three scheduling algorithms.

Figs. 7 and 8 present the energy performance of the four studied scheduling algorithms when executing different numbers of tasks in an edge computing environment. In this experiment, the number of VMs on the edge servers was set to 10. The energy consumption index of the edge server was obtained from work reported in [34]. The energy consumption reported here refers to only the CPU resources consumed during task execution. Details on the energy consumption model can be found in [32].

Figs. 7 and 8 depict the energy performance of our proposed EA-DFPSO, SJF, FCFS and Round Robin scheduling algorithms in terms of the energy consumed while executing respective tasks. The number of tasks (X-axis) is plotted against the energy consumption (Y-axis) for the four scheduling algorithms. Every set of experiments was repeated 10 times to enhance the accuracy and reliability of the results. As the number of tasks increased from 40 to 160, the edge servers were characterized by obvious energy consumption increases (Figs. 7 and 8). Despite this, our proposed algorithm consistently provided the optimal energy consumption performance, consuming 3.26 J, 4.58 J, 4.88 J, and 5.54 J energy when executing 40, 80, 120 and 160 tasks, respectively. This is due to the fact that our proposed EA-DFPSO algorithm utilizes the energy-aware fitness function during task scheduling. In our proposed EA-DFPSO algorithm, two fitness functions are used to search for the best particle, ensuring the best possible scheduling scheme was selected to save energy and reduce task completion times. Given that the traditional PSO algorithm uses a single fitness function, the dual fitness function used in our proposed EA-DFPSO algorithm helps to enhance energy and time performances significantly. As discussed in Section 3.7, this dual fitness function contributed to an obvious performance improvement in our proposed EA-DFPSO algorithm, thus outperforming other algorithms.

To demonstrate the energy superiority of our EA-DFPSO algorithm more intuitively, we conducted more experiments, as shown in Figs. 9–12.

Fig. 9 illustrates the performance of our proposed EA-DFPSO algorithm and the SJF algorithm while scheduling four sets of tasks ranging from 40 to 160. Our proposed EA-DFPSO algorithm exhibited better energy efficiency than SJF, with the proportion of energy conservation witnessed at 8.30% for 40 tasks, 8.80% for 80 tasks, 4.66% for 120 tasks, and 8.10% for 160 tasks. In an ideal scenario, increasing the number of tasks should cause the proportion of energy conserved by EA-DFPSO to increase gradually against SJF. However, this trend is not reflected in Fig. 9, which we attribute to the random task lengths generated for the experiments. The SJF scheduling algorithm was conducive to the priority execution of short tasks.

Fig. 10 presents the performance of our proposed EA-DFPSO algorithm and the FCFS algorithm for task groups numbering 40 to 160, with the former exhibiting energy conservation against the latter of 27.13% for 40 tasks, 7.96% for 80 tasks, 17.97% for 120 tasks, and 13.18% for 160 tasks, respectively.

Fig. 11 presents the performance of our proposed EA-DFPSO algorithm and the Round Robin algorithm when scheduling four sets of tasks of different numbers from 40 through to 160. Our EA-DFPSO algorithm exhibited much better energy efficiency than the Round Robin algorithm, with the proportion of energy conservation witnessed at 19.58% for 40 tasks, 17.47% for 80 tasks, 17.84% for 120 tasks and 17.95% for 160 tasks, respectively. The Round Robin algorithm is a relatively fair scheduling algorithm, as it creates nearly similar execution slots for all tasks. But this feature can also delay the completion of long-running tasks.

Fig. 12 depicts the average proportional energy conservation performance of our proposed EA-DFPSO algorithm against the studied SJF, FCFS, and Round Robin algorithms. It can be observed that our proposed EA-DFPSO algorithm saved 8.09%, 16.56% and 17.99% of the energy than the SJF, FCFS and Round Robin algorithms, respectively, in an edge computing environment.

To further evaluate the performance of our proposed EA-DFPSO model, we conducted more comparisons with some other state-of-the-art algorithms [35], such as Bat Swarm (BA), Improved Chaotic Bat Swarm (ICBA), and Quantum-behaved PSO (QPSO). Table 3 expresses task completion times and energy consumption as T and E, respectively. These four scheduling algorithms were used to schedule into 10 VMs. Through four sets of experiments with different numbers of tasks, it can

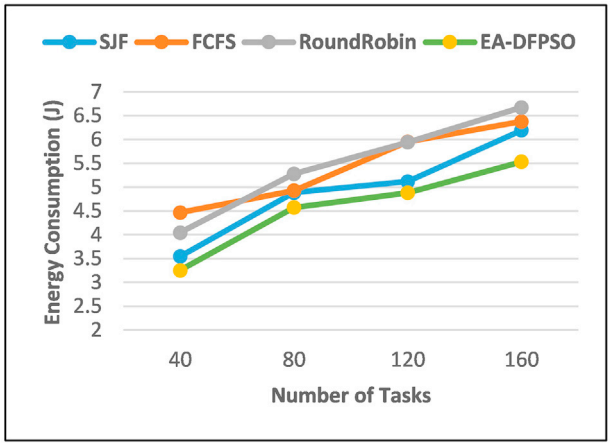

Fig. 7. Energy consumption with different numbers of tasks.

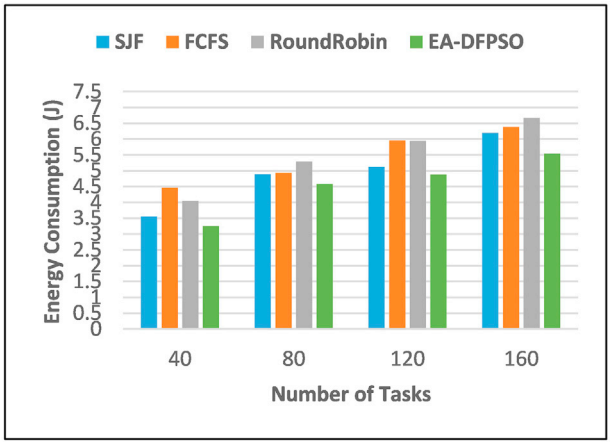

Fig. 8. Energy consumption with different numbers of tasks.

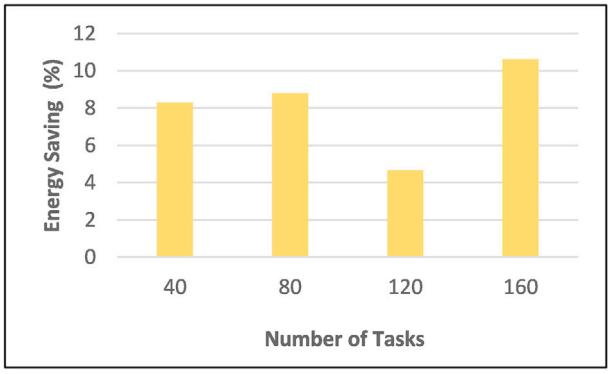

Fig. 9. Comparing EA-DFPSO with SJF

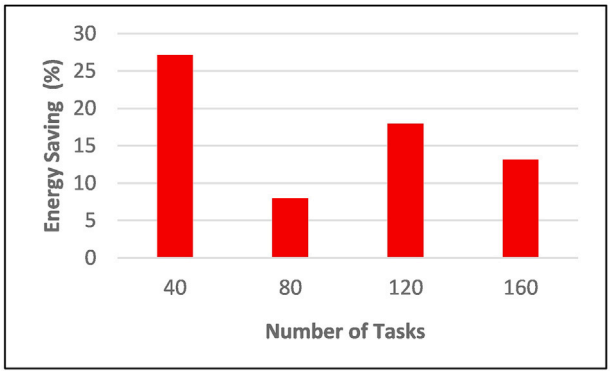

Fig. 10. Comparing EA-DFPSO with FCFS.

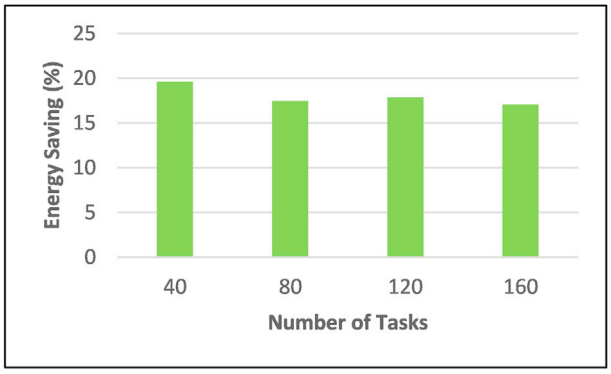

Fig. 11. Comparing EA-DFPSO with RoundRobin.

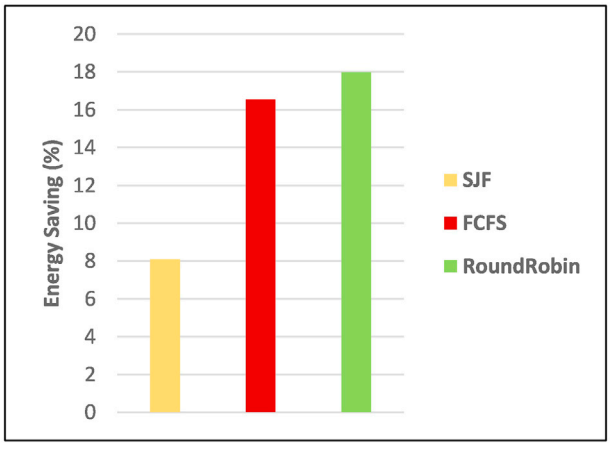

Fig. 12. Average energy saving of EA-DFPSO compared with SJF, FCFS and RoundRobin.

be seen that our proposed model EA-DFPSO always consumed minimal energy when compared with the other three algorithms. This is because our model adopted double fitness functions to search for the optimal scheduling scheme, unlike the other three algorithms, which adopted completion time as the only decision-making standard. Although some cases of ICBA and QPSO can achieve completion time similar to our model, they consumed more energy than our EA-DFPSO model in an edge environment, proving our model's superior energy efficiency performance in edge servers.

6. Conclusions

The rapid growth of IoT devices and the acceleration of commercializing 5G technology is introducing tremendous pressure on communication networks and traditional cloud data centers. Furthermore, traditional cloud architecture has proven unsuitable for latency-sensitive applications by being located far away from the mobile terminals. Edge computing is proving to be a promising solution for such latency-

Table 3
Algorithm comparisons.

MODELS		B A		ICBA		QPSO		EA-DFPSO	
TASKS	VMS	T	E	T	E	T	E	T	E
40	10	27.68	4.53	20.31	3.97	20.35	3.79	20.91	**3.26**
80	10	37.40	7.45	27.13	6.21	29.42	5.96	28.81	**4.58**
120	10	41.38	8.91	33.26	6.83	30.15	6.65	32.94	**4.89**
160	10	53.73	9.84	40.42	7.60	40.58	7.46	40.29	**5.50**

T means Time (S); E means Energy (J).

sensitive applications, hosting task execution at the edge. However, edge computing also requires an optimized scheduling strategy to allocate tasks among edge nodes for faster completion and energy conservation. This paper proposed a novel task scheduling algorithm for achieving energy efficiency in an edge computing environment, named EA-DFPSO, based on an improved PSO algorithm. The proposed algorithm applies double fitness functions to search for an optimal tasks scheduling scheme for saving energy in edge servers. Meanwhile, an optimized inertia weight was used in our proposed scheduling algorithm, which helps EA-DFPSO achieve the best performance. Extensive experimentation indicated that the proposed EA-DFPSO could shorten the task completion time and consume less energy than the SJF, FCFS, Round Robin, BA, ICBA, and QPSO algorithms. We plan to develop a novel energy-efficient scheduling model that collaborates and coordinates task execution among cloud and edge computing in our future work.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was partially supported by UK-Jiangsu 20-20 World Class University Initiative programme.

References

[1] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, D.O. Wu, Edge Computing in Industrial Internet of Things: Architecture, Advances and Challenges, IEEE Communications Surveys & Tutorials, 2020.
[2] Media, Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017– 2022. http://media.mediapost.com/uploads/CiscoForecast.pdf, 2019 (accessed 8 December 2021)
[3] W. Yu, et al., A survey on the edge computing for the Internet of Things, IEEE access 6 (2017) 6900–6919.
[4] M. Long, F. Peng, Y. Zhu, Identifying natural images and computer generated graphics based on binary similarity measures of PRNU, Multimed. Tool. Appl. 78 (1) (2019) 489–506.
[5] B. Li, J. Li, L. Liu, CloudMon: a resource-efficient IaaS cloud monitoring system based on networked intrusion detection system virtual appliances, Concurrency Comput. Pract. Ex. 27 (8) (2015) 1861–1885.
[6] H. Al-Aqrabi, L. Liu, R. Hill, N. Antonopoulos, Taking the business intelligence to the clouds, in: 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems, IEEE, 2012, pp. 953–958.
[7] Y. Lu, L. Liu, J. Panneerselvam, X. Zhai, X. Sun, N. Antonopoulos, Latency-based Analytic Approach to Forecast Cloud Workload Trend for Sustainable Datacentres, IEEE Transactions on Sustainable Computing, 2019.
[8] L. Gu, J. Cai, D. Zeng, Y. Zhang, H. Jin, W. Dai, Energy efficient task allocation and energy scheduling in green energy powered edge computing, Future Generat. Comput. Syst. 95 (2019) 89–99.
[9] M. Caprolu, R. Di Pietro, F. Lombardi, S. Raponi, Edge computing perspectives: architectures, technologies, and open security issues, in: 2019 IEEE International Conference on Edge Computing (EDGE), IEEE, 2019, pp. 116–123.
[10] Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: the communication perspective, IEEE Communications Surveys & Tutorials 19 (4) (2017) 2322–2358.
[11] M. Bouet, V. Conan, Mobile edge computing resources optimization: a geo-clustering approach, IEEE Transactions on Network and Service Management 15 (2) (2018) 787–796.
[12] Y. Lu, L. Liu, J. Panneerselvam, B. Yuan, J. Gu, N. Antonopoulos, A GRU-based prediction framework for intelligent resource management at cloud data centres in the age of 5G, IEEE Transactions on Cognitive Communications and Networking 6 (2) (2020) 486–498, https://doi.org/10.1109/TCCN.2019.2954388.
[13] P. Wang, C. Yao, Z. Zheng, G. Sun, L. Song, Joint task assignment, transmission, and computing resource allocation in multilayer mobile edge computing systems, IEEE Internet of Things Journal 6 (2) (2018) 2872–2884.
[14] B.M. Nguyen, H. Thi Thanh Binh, B. Do Son, Evolutionary algorithms to optimize task scheduling problem for the IoT based bag-of-tasks application in cloud–fog computing environment, Appl. Sci. 9 (9) (2019) 1730.
[15] X. Niu, et al., Workload allocation mechanism for minimum service delay in edge computing-based power Internet of Things, IEEE Access 7 (2019) 83771–83784.
[16] Y. Wu, K. Ni, C. Zhang, L.P. Qian, D.H. Tsang, NOMA-assisted multi-access mobile edge computing: a joint optimization of computation offloading and time allocation, IEEE Trans. Veh. Technol. 67 (12) (2018) 12244–12258.
[17] G. Guo, J. Zhang, Energy-efficient incremental offloading of neural network computations in mobile edge computing, in: GLOBECOM 2020-2020 IEEE Global Communications Conference, IEEE, 2020, pp. 1–6.
[18] S. Amiri, M. Hosseinabady, A. Rodriguez, R. Asenjo, A. Navarro, J. Nunez-Yanez, Workload partitioning strategy for improved parallelism on FPGA-CPU heterogeneous chips, in: 2018 28th International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2018, pp. 376–3764.
[19] P.K. Muhuri, P.K. Gupta, J.M. Mendel, User-satisfaction-aware power management in mobile devices based on perceptual computing, IEEE Trans. Fuzzy Syst. 26 (4) (2017) 2311–2323.
[20] Q. Zhang, M. Lin, L.T. Yang, Z. Chen, S.U. Khan, P. Li, A double deep Q-learning model for energy-efficient edge scheduling, IEEE Transactions on Services Computing 12 (5) (2018) 739–749.
[21] G. Jia, G. Han, J. Du, S. Chan, A maximum cache value policy in hybrid memory-based edge computing for mobile devices, IEEE Internet of Things Journal 6 (3) (2018) 4401–4410.
[22] W. Zhang, Y. Liu, G. Han, Y. Feng, Y. Zhao, An energy efficient and QoS aware routing algorithm based on data classification for industrial wireless sensor networks, IEEE Access 6 (2018) 46495–46504.
[23] Y. Dong, S. Guo, J. Liu, Y. Yang, Energy-efficient fair cooperation fog computing in mobile edge networks for smart city, IEEE Internet of Things Journal 6 (5) (2019) 7543–7554.
[24] D. Zhang, Y. Ma, C. Zheng, Y. Zhang, X.S. Hu, D. Wang, Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing, in: 2018 IEEE/ACM Symposium on Edge Computing (SEC), IEEE, 2018, pp. 243–259.
[25] K. Ha, et al., You can teach elephants to dance: agile VM handoff for edge computing, in: Proceedings of the Second ACM/IEEE Symposium on Edge Computing, 2017, pp. 1–14.
[26] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, X.S. Shen, Energy Efficient Dynamic Offloading in Mobile Edge Computing for Internet of Things, IEEE Transactions on Cloud Computing, 2019.
[27] F. Marini, B. Walczak, Particle swarm optimization (PSO). A tutorial, Chemometr. Intell. Lab. Syst. 149 (2015) 153–165.
[28] T.F. Congfeng Jiang, Honghao Gao, Weisong Shi, Liangkai Liu, Christophe Cérin, Jian Wan, Energy aware edge computing: a survey, Comput. Commun. 151 (2020) 556–580.
[29] H. Ali, U.U. Tariq, Y. Zheng, X. Zhai, L. Liu, Contention & energy-aware real-time task mapping on NoC based heterogeneous MPSoCs, IEEE Access 6 (2018) 75110–75123.
[30] GitHub, CloudSim 4.0. https://github.com/Cloudslab/cloudsim/releases/tag/cloudsim-4.0, 2016 (accessed 2 December 201)
[31] J. Ru, J. Keung, An empirical investigation on the simulation of priority and shortest-job-first scheduling for cloud-based software systems, in: 2013 22nd Australian Software Engineering Conference, IEEE, 2013, pp. 78–87.
[32] A. Karthick, E. Ramaraj, R.G. Subramanian, An efficient multi queue job scheduling for cloud computing, in: 2014 World Congress on Computing and Communication Technologies, IEEE, 2014, pp. 164–166.
[33] A. Singh, P. Goyal, S. Batra, An optimized round robin scheduling algorithm for CPU scheduling, Int. J. Comput. Sci. Eng. 2 (7) (2010) 2383–2385.
[34] NOTEBOOKCHECK, Benchmarks/Tech. https://www.notebookcheck.net/Intel-Core-i7-8550U-SoC-Benchmarks-and-Specs.242108.0.html, 2017 (accessed 7 May 2021)
[35] C. Jian, J. Chen, J. Ping, M. Zhang, An improved chaotic bat swarm scheduling learning model on edge computing, IEEE Access 7 (2019) 58602–58610.