IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2022.DOI

A Particle Swarm Optimization with Lévy Flight for Service Caching and Task Offloading in Edge-Cloud Computing

TIELIANG GAO[1], QIGUI TANG[1], JIAO LI[1], YI ZHANG[1], YIQIU LI[1], JINGYA ZHANG[1]
[1]Key Laboratory of Data Analysis and Financial Risk Prediction, Xinxiang University, Xinxiang, 453003, China

The research was supported in part by the National Natural Science Foundation of China (Grant No. 61902021, 61975187, 62072414), the Key Science and Technology Program of Henan Province (Grant No. 222102210218, 212102210096, 212102210104), and the Soft Science Research Project of Henan Province (Grant No. 222400410137).

ABSTRACT Edge-cloud computing is an efficient approach to address the high latency issue in mobile cloud computing for service provisioning, by placing several computing resources close to end devices. To improve the user satisfaction and the resource efficiency, this paper focuses on the task offloading and service caching problem for providing services by edge-cloud computing. This paper formulates the problem as a constrained discrete optimization problem, and proposes a hybrid heuristic method based on Particle Swarm Optimization (PSO) to solve the problem in polynomial time. The proposed method, LMPSO, exploit PSO to solve the service caching problem. To avoid PSO trapping into local optimization, LMPSO adds lévy flight movement for particle updating to improve the diversity of particle. Given the service caching solution, LMPSO uses a heuristic method with three stages for task offloading, where the first stage tries to make full use of cloud resources, the second stage uses edge resources for satisfying requirements of latency-sensitive tasks, and the last stage improves the overall performance of task executions by re-offloaded some tasks from the cloud to edges. Simulated experiment results show that LMPSO has upto 156% better user satisfaction, upto 57.9% higher resource efficiency, and upto 155% greater processing efficiency, in overall, compared with other seven heuristic and meta-heuristic methods.

INDEX TERMS Edge-Cloud Computing, Particle Swarm Optimization, Service Caching, Task Offloading

I. INTRODUCTION

NOWADAYS, mobile and Internet of Thing (IoT) devices can be seen everywhere, and their popularity is on the rise with the increasing of user requirements and the development of information and communication technology [1]–[3]. As shown in Cisco Annual Internet Report [4], global mobile subscribers will account for 71% of population, and IoT devices will account for about half of global networked devices, by 2023. The mobile and IoT users' requirements cannot be guaranteed only by cloud computing for service delivery, due to the abundant variety and quantity of Internet services and the high network latency of the cloud [5]. Therefore, more and more service providers use edge computing to improve the service quality, by placing some computing resources close to user devices [6].

Due to the restricted space of edge computing centres (edges for short), there are limited amount of computing and storage resources in edges. Therefore, edge computing cannot provide all services at a time. Thus, edge-cloud computing is an efficient way to deliver both latency-sensitive and resource-hungry services, by combining the advantages of both edge computing and cloud computing [7]. In edge-cloud computing, some services are deployed (cached) on an edge for latency-sensitive requests, and the cloud can deliver all services due to its "infinite" computing and storage resources.

There are two decisions must be made for service provisioning by edge-cloud computing with high efficiency and performance, service caching and task offloading. The service caching is to decide which services are deployed on edges. Task offloading decides the location where each request task is served. These two decision-making problems are interrelated. A task can be offloaded to an edge only when its requested service is cached on the edge. In general, it is better to cache services with more requests and offload more requests into edges, as edges provide much lower network latency than the cloud. But due to the limited resources on

edges, offloading more tasks to edges can result in an increasing of the computing latency. Thus, the service caching and task offloading strategies should be jointly designed carefully for balancing the computing latency on edges and the network latency on the cloud to improve the service quality.

Unfortunately, the joint service caching and task offloading problem is NP hard [8]. There mainly three kinds of methods to address the problem, which are respectively heuristics, meta-heuristics, and machine learning (ML). In general, heuristics are is fast to provide solutions, but the solution performance is limited due to their local search ideas. Meta-heuristics can achieve better solutions than heuristics, benefiting from their global search abilities, but they usually consume more computing time [9]. ML-based methods learn some patterns from the historical states of task executions, and make caching and offloading decisions for subsequent tasks based on learned patterns. ML-based methods may achieve better performance than other two kinds of methods, but they are too costly for the pattern learning (training).

Therefore, several researches used meta-heuristics, e.g., Genetic Algorithm (GA) [10], Particle Swarm Optimization (PSO) [11], Whale Optimization Algorithm (WOA) [12], and Cuckoo Search Algorithm (CSA) [13], for solving the service caching and task offloading problem. This is mainly because meta-heuristics usually achieve a better performance than heuristics due to their global search abilities. Some works exploited the hybrid of two meta-heuristics, e.g., PSO + GA [14], which can achieve a better performance than each one. While, these methods didn't consider to integrate heuristic local search strategy into a meta-heuristic for the joint service caching and task offloading problem, even though it have been used in other problems due to the good performance by combining both advantages of heuristics and meta-heuristics [15], [16].

Thus, this paper exploits a hybrid heuristic method by combining a representative meta-heuristic algorithm, Particle Swarm Optimization (PSO), and a heuristic algorithm for joint service caching and task offloading. Specifically, the proposed method, named LMPSO, employs PSO to achieve a service caching, with the fitness evaluated by the user satisfaction and resource efficiency. To achieve a task offloading solution given a service caching solution, LMPSO uses a heuristic algorithm to improve the user satisfaction and resource efficiency. To overcome the issue that PSO is easily trapped into local optimization, LMPSO integrates lévy flight scheme into particle movements to improve the the diversity of particles. The contributions can be summarized as follows.

1) The joint service caching and task offloading problem is modelled into a constrained discrete optimization problem, to minimize the user satisfaction and resource efficiency, for edge-cloud computing. In this paper, the user satisfaction is quantified by the number of tasks with the satisfaction of their requirements. The resource efficiency is evaluated by the overall resource utilization.
2) To solve the joint service caching and task offloading problem in polynomial time, A hybrid heuristic algorithm is designed, which is named as LMPSO. LMPSO exploits PSO algorithm, and employs lévy flight in particle moving to improve the performance of PSO. In LMPSO, each particle position represents a service caching solution, and the value in a dimension is the edge servers that the corresponding service is cached. Given a service caching solution, LMPSO uses a heuristic method with three stages for task offloading. The first stage is pre-offloading as many tasks as possible to the cloud, for making full use of the richness of cloud resources. The second stage offloads remain tasks to edges for executing latency-sensitive tasks. At last stage, the heuristic method re-offloads tasks from the cloud to edges to improve the overall performance of task execution.
3) Extensive simulated experiments are conducted for evaluating the performance of LMPSO, where parameters are set referring to existing works and the reality. The experiment results show that LMPSO has 8.34%–156%, 1.00%–57.9%, and 7.91%–155% better performance in user satisfaction, resource efficiency, and processing efficiency, respectively, compared with other seven heuristic and meta-heuristic methods, in overall.

The rest of this paper was organized as follows. Section II models the joint service caching and task offloading problem concerned in this paper, and Section III proposes the heuristic algorithm to address the problem optimizing the user satisfaction and resource efficiency. Section IV evaluates the proposed heuristic algorithm by conducting simulated experiments. Section V discusses the research works. And finally, Section VI concludes the paper.

II. PROBLEM STATEMENT

A. SYSTEM MODEL

This paper considers an edge-cloud computing system consisting of multiple edges and one cloud, as shown in Fig. 1. Each edge is equipped with one or more edge servers (ESs). There are multiple user devices having communication connections with an edge, i.e., these devices are covered by the edge. A user launches its requests by its device, and these requests can be only processed by the cloud and the edge covering the device. The cloud provides various types of cloud servers (CSs). For each service, it can process request tasks on an ES only when the service is cached on the ES.

In the system, there are E ESs. For optimizing the user satisfaction, V CSs are rented from the cloud. $n_j,\ 1 \leq j \leq E+V$ are used to represent these ESs and CSs, where $n_j,\ 1 \leq j \leq E$ are ESs. For ES/CS n_j, it has c_j^c computing capacity, c_j^s storage space, and l_j communication channels, where each channel has b_j^d/b_j^u downlink/uplink transmission capacity. The cloud can provide all kinds of services due to its limitless storage space. Thus, for CSs, $n_j,\ E+1 \leq j \leq E+V$, their storage spaces are set as

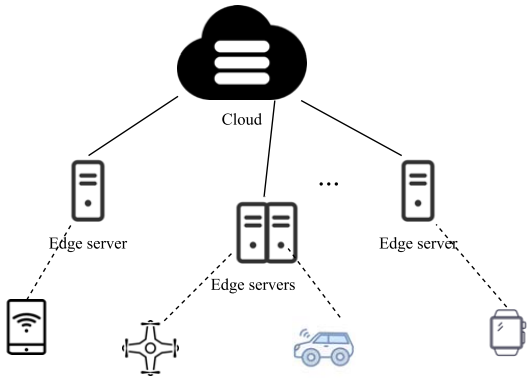


FIGURE 1. The edge-cloud computing system

infinite, i.e., $c_j^s = +\infty,\ E+1 \le j \le E+V$. Usually, the cloud provider equips each CS with one Ethernet card. Thus, the number of communication channels of a CS is one, i.e., $l_j = 1,\ E+1 \le j \le E+V$.

Provider delivers S services ($a_k,\ 1 \le k \le S$) for users in the edge-cloud computing system. For the service a_k, it needs s_k storage space for its deployment. To represent the service caching solution, binary variables, $x_{j,k},\ 1 \le j \le E, 1 \le k \le S$, are defined to indicate whether a service is cached on an ES. $x_{j,k} = 1$ if service a_k is cached on ES n_j, and $x_{j,k} = 0$ if not.

There are T request tasks ($t_i,\ 1 \le i \le T$) needed to be served in the system. The service required by t_i is a_{r_i}. For task t_i, it requires h_i computing size for its completion. The input and output data amounts of t_i are g_i^d and g_i^u, respectively. This research concerns hard deadline tasks. Task t_i must be finished within its deadline (d_i). If a task cannot be finished within its deadline, the service provider has no profit for processing the task, and thus rejects the task's request. Binary variables $y_{i,j},\ 1 \le i \le T, 1 \le j \le E+V$ are used to represent the task offloading solution. $y_{i,j} = 1$ means t_i is offloaded to ES/CS n_j, and $y_{i,j} = 0$ means not. When a task is offloaded to edges, it only can be offloaded to ESs that its requested service is cached on. Thus,

$$y_{i,j} \le x_{j,r_i},\ 1 \le i \le T, 1 \le j \le E. \tag{1}$$

B. TASK EXECUTION MODEL

When task t_i is offloaded to ES/CS n_j, i.e. $y_{i,j} = 1$, the time consumed by the transitions of its input and output data are g_i^d/b_i^d and g_i^u/b_i^u, respectively. The ES/CS needs h_i/c_j^c computing time for processing the task's input data to achieve the output data. When multiple tasks are offloaded to one computing node (an ES or a CS), they compete for the network and computing resources on the node. In real world, the output data amount is much less than the input data amount for a task. Therefore, there are two situations for a node processing multiple tasks, which are that the bottlenecks are network and computing resources, respectively. In the first situation, the computing of a task can be started only after finishing its input data transmission. Thus, the finish time of a task can be calculated by

$$ft_i^{net} = \sum_{j=1}^{E+V} (y_{i,j} \cdot (ftn_i + h_i/c_j^c + g_i^u/b_j^u)), \\ 1 \le i \le T, \tag{2}$$

and

$$ftn_i = \sum_{j=1}^{E+V} (y_{i,j} \cdot (stn_i + g_i^d/b_j^d)), 1 \le i \le T, \tag{3}$$

where ft_i^{net} is the finish time of task t_i when the network resource is bottleneck, and ftn_i and stn_i is the finish and start time of the input data transmission of t_i, respectively. The network resource is available for transferring the input data of t_i only when all input data transmissions are finished for all tasks that are allocated to the same communication channel and started before t_i. Thus, the start time of the input data transmission for a task satisfies

$$z_{i,j,m} \cdot stn_i \ge \max_{o_{ii}<o_i} \{z_{ii,j,m} \cdot ftn_{ii}\}, \\ 1 \le i \le T, 1 \le j \le E+V, 1 \le m \le l_j, \tag{4}$$

where $o_i \in \{1, 2, ..., T\}$ is used to indicate the processing order of tasks offloaded to a computing node. $o_{ii} < o_i$ means t_{ii} is processed before t_i. $z_{i,j,m}$ is a binary variable to indicate the communication channel allocation for t_i. $z_{i,j,m} = 1$ if t_i is offloaded to n_j and the mth channel is allocated to the task for its input data transmission, and otherwise $z_{i,j,m} = 0$. Then Eq. (5) are satisfied. As two tasks cannot be processed simultaneously on a channel or a computing node, Eq. (6) must hold.

$$y_{i,j} = \sum_{m=1}^{l_j} z_{i,j,m}, 1 \le i \le T, 1 \le j \le E+V. \tag{5}$$

$$(i \ne ii) \rightarrow (o_i \ne o_{ii}), 1 \le i, ii \le T. \tag{6}$$

In the second situation that the computing resources is the bottleneck, when t_i is offloaded to n_j, its input data transfer has been complete before the computing resource is available to it. In this case, the computing time of a task is started when its previous task completes its computing. Thus, the finish time of each task can be calculated by

$$ft_i^{cpu} = \sum_{j=1}^{E+V} (y_{i,j} \cdot (\max_{o_{ii}<o_i} \{ft_{ii} - g_{ii}^u/b_j^u\} \\ + h_i/c_j^c + g_i^u/b_j^u)), 1 \le i \le T, \tag{7}$$

where ft_i is the finish time of task t_i, i.e., the time when its output data transfer is complete. $ft_i - g_i^u/b_j^u$ is the finish time of t_i's computing when it is offloaded to n_j.

Combining these two situations, the finish time of tasks can be achieved by

$$ft_i = \max\{ft_i^{net}, ft_i^{cpu}\}, 1 \le i \le T. \tag{8}$$

And the deadline constraints can be formulated as

$$ft_i \leq d_i, 1 \leq i \leq T. \tag{9}$$

Noticing that $ft_i = 0$ if t_i is not offloaded to any computing node, based on Eq. (2), (7), and (8). Thus, Eq. (9) also hold for rejected tasks.

For a computing node, it is occupied until all tasks that offloaded to the node are finished, thus the amount of occupied computing resources for ESs are respectively

$$oc_j = c_j^c \cdot \max_{i=1}^{T}\{y_{i,j} \cdot ft_i\}, 1 \leq j \leq E. \tag{10}$$

But CSs charged in unit time, e.g., hour, in most of cloud providers. The actual occupied time need to be rounded up. Therefore, the occupied computing resources of CSs are respectively

$$oc_j = c_j^c \cdot \max_{i=1}^{T}\{\lceil y_{i,j} \cdot ft_i \rceil\}, E+1 \leq j \leq E+V. \tag{11}$$

In each ES/CS, the amount of computing resources actually used for computing tasks is the accumulated size of computing size required by tasks offloaded to the node, i.e.,

$$uc_j = \sum_{i=1}^{T}(y_{i,j} \cdot h_i), 1 \leq j \leq E+V. \tag{12}$$

Thus, the computing resource utilization of each computing node can be calculated by Eq. (13), and the overall computing resource utilization in the system can be achieved by Eq. (14).

$$u_j = uc_j/oc_j, 1 \leq j \leq E+V. \tag{13}$$

$$U = \frac{\sum_{j=1}^{E+V} uc_j}{\sum_{j=1}^{E+V} oc_j}. \tag{14}$$

In the edge-cloud computing system, a task can be offloaded to only one computing node, thus, Eq. (15) hold. In this paper, task redundancy execution is not considered for performance improvement, as the redundancy consumes extra resources, which results in a low resource efficiency.

$$\sum_{j=1}^{E+V} y_{i,j} \leq 1,\ 1 \leq i \leq T. \tag{15}$$

Then, the number of tasks accepted by the system for their processing is

$$N = \sum_{i=1}^{T}\sum_{j=1}^{E+V} y_{i,j}. \tag{16}$$

C. PROBLEM MODEL

This paper considers to optimize the user satisfaction and the resource efficiency. The number of accepted tasks and the computing resource utilization are used as the metrics for quantifying these two optimization objectives, respectively. The formulated model is also compatible with other metrics. Based on previous formulations, the joint service caching and task offloading problem can be modelled into

$$\text{maximizing}\ \ N + U, \tag{17}$$

subject to:

$$Eq.(1)\text{–}(16), \tag{18}$$

$$x_{j,k} \in \{0,1\}, 1 \leq j \leq E, 1 \leq k \leq S, \tag{19}$$

$$z_{i,j,m} \in \{0,1\}, \quad 1 \leq i \leq T, 1 \leq j \leq E+V, 1 \leq m \leq l_j, \tag{20}$$

$$o_i \in \{1,2,...,T\}, 1 \leq i \leq T.$$

Noticing that the utilization U is less than 1, the main objective is optimizing the user satisfaction. Decision variables include $x_{j,k}, 1 \leq j \leq E, 1 \leq k \leq S$, $z_{i,j,m}, 1 \leq i \leq T, 1 \leq j \leq E+V, 1 \leq m \leq l_j$, and $o_i, 1 \leq i \leq T$. $x_{j,k}$ indicates the service caching solution. $z_{i,j,m}$ represents the solutions of the task offloading and the communication channel allocation. o_i denotes the task execution order in each computing node. Due to the discreteness of decision variables, this optimization problem is hard to solve. Therefore, a polynomial time method is proposed to solve the problem, as illustrated in the following section.

III. HYBRID HEURISTIC JOINT SERVICE CACHING AND TASK OFFLOADING METHOD

This section presents LMPSO, the joint service caching and task offloading method, based on PSO, in details. Similar to PSO, LMPSO initializes the position of each particle randomly, and repeats following three steps: 1) evaluating the fitness for each particle, and updating the global best position and local positions, 2) updating the position of each particle as done in PSO, 3) updating the position of each particle with Lévy Flight. Next, the encoding/decoding method used by LMPSO is illustrated first, which is the transform method between a particle position and the service caching solution. Then, three repeated steps of LMPSO are detailed, respectively.

A. ENCODING METHOD

LMPSO uses particle positions to represent service caching solution. There a one-to-one relationship between services and dimensions of each particle position. For caching S services on E ESs, there are S dimensions for each particle position, where the position on a dimension represents ESs that the corresponding service is cached on. For each dimension, the position value is integer, and its range is [0, 2^E], where the jth bit is indicating whether the service is cached on ES n_j, by convert the position value to binary. For example, as shown in Fig. 2, if there are 5 services $(s_1, s_2, ..., s_5)$ and 5 ESs $(n_1, n_2, ..., n_5)$, the number of dimensions for each particle position is 10. A position of (2, 3, 5, 20, 30) means that s_1 is cached on n_2 ($(2)_{10}$=$(00010)_2$), s_2 is cached on n_1 and n_2, s_3 is cached on n_1 and n_3, s_4 is cached on n_3 and n_5, and s_5 is cached on n_2, n_3, n_4 and n_5.

B. STEP 1: FITNESS EVALUATION

The fitness of a particle is the optimization objective (17) value of the joint service caching and task offloading solution derived from its position. Based on above encoding/decoding

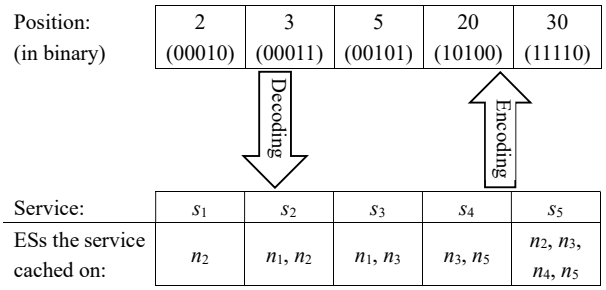


FIGURE 2. An example of encoding/decoding method

method, the position can be decoded into the service caching solution for each particle. Given the service caching solution, LMPSO exploits a heuristic task offloading method with following three stages.

In the first stage, for tasks whose requirements can be satisfied by the cloud, LMPSO pre-offloads them to the cloud. In every time a new CS needed to be pre-rented, LMPSO chooses the CS type with the best cost-performance ratio, for optimizing the cost efficiency on the cloud. In this stage, the abundance of cloud resources can be made full use for satisfying requirements of network latency-insensitive tasks.

In the second stage, for tasks that cannot be finished on the cloud, LMPSO offloads them to ESs, with the service caching solution decoded by the particle position. In this stage, edge resources with low network delay are exploited for processing latency-sensitive tasks.

In the last stage, to improving the resource usage and the task performance, LMPSO re-offloads tasks from CSs to ESs. LMPSO exams each task pre-offloaded to the cloud. If the task can be finished by an ES, then, LMPSO re-offloaded it to the ES. In the end of the stage, for each CS, if all tasks that are pre-offloaded to are re-offloaded to ESs, LMPSO cancels the pre-rent of the CS.

Within each of these three stages, one of the simplest scheduling method, First Fit (FF), is used to decide the computing node, the communication channel, and the execution order for each task. Any other scheduling method can be easily integrated with LMPSO. The design of more efficient scheduling methods is one of our future works.

Now, given a particle, a joint service caching and task offloading solution can be gotten from its position. Based on the solution, the number of accepted tasks, N, and the overall computing resource utilization, U, can be calculated according to Eq.(1)–(16). And thus, the fitness value can be achieved for each particle.

C. STEP 2: PARTICLE MOVEMENT OF PSO

LMPSO conducts two kinds of movement for the position updating of each particle, the movements of PSO and the lévy flight scheme, which are respectively illustrated in this subsection and the next subsection.

In PSO, the position of each particle is updated for moving it towards its local best position and the global position with a certain inertia. The updating of a particle in each iteration step in PSO is as following.

$$vel_k = \omega \cdot vel_k + r_1 \cdot acc_1 \cdot (lbest_i - p_k) + r_2 \cdot acc_2 \cdot (gbest_i - p_k), \quad (21)$$

$$p_k = p_k + vel_k, \quad k = 1, 2, ..., S, \quad (22)$$

where vel_k is the movement velocity in the kth dimension for the particle. p_k is the position in the kth dimension. ω is the inertia weight of the particle. acc_1 and acc_2 are acceleration coefficients for moving towards the local best position and the global position, respectively. r_1 and r_2 are two random numbers between 0 and 1.

D. STEP 3: PARTICLE MOVEMENT WITH LÉVY FLIGHT

Because PSO is easily trapped into a local best solution, the lévy flight movement is added for updating the particle positions, to improve the diversity of particles. In each iteration step, for every particle, after updating the position using Eq. (21) and (21), LMPSO executes additional random movement with a certain probability, based on lévy flight. The additional updating of the position for each particle is shown in Eq. (23).

$$p_k = \begin{cases} p_k + levy \cdot (r_1 \cdot acc_1 \cdot (lbest_i - p_k) + r_2 \cdot acc_2 \cdot (gbest_i - p_k), & \text{if r < R} \\ p_k, & \text{else} \end{cases} \quad k = 1, 2, ..., S. \quad (23)$$

Where r is a random value in the range of [0, 1]. R is a constant to indicate the probability of adding the lévy flight movement for the particle's updating. $levy$ is a random value following lévy flight distribution which is $\mu/|\nu|^{-\beta}$. μ and ν are random variables following Gaussian distributions with zero mean. The standard deviation of ν is 1, and that of μ is

$$\frac{\Gamma(1+\beta) \cdot sin(\frac{\pi \cdot \beta}{2})}{\beta \cdot \Gamma(1+\frac{1+\beta}{2}) \cdot 2^{\frac{\beta-1}{2}}}, \quad (24)$$

where $\Gamma(\bullet)$ is the Gamma function. β is the parameter of the lévy flight distribution. In this paper, referring to [17], β is set between 1 and 3 randomly.

IV. PERFORMANCE EVALUATION

In this section, the performance of LMPSO is evaluated by conducting extensive simulated experiments, and the experiment results are analysed.

A. EXPERIMENT DESIGN

In the simulated edge-cloud system, there are 1000 tasks randomly requesting one of 100 services. For every task, its required computing size is randomly set between 0.5–1.2GHz, and the amount of its input data is in the range of [5, 6]MB, referring to [18]. The deadline of each task is set between 1 and 5 seconds. Each service requires [40, 80]GB storage space for its deployment, referring to [19].

The system includes 10 ESs and one CS type. Each ES is configured with 20GHz computing capacity, 300GB storage space, and 10 communication channels each with 60Mbps transmission bandwidth. The configuration of the CS type is 5GHz computing capacity and 15Mbps. The parameters of ESs and the CS type are set referring to [19].

LMPSO is compared with five heuristic methods, FF, FFD (First Fit Decreasing), EDF (Earliest Deadline First), PSC (Popularity-based Service Caching) [20], CEDC-A (Approximation algorithm for Constrained Edge Data Caching) [21], and two meta-heuristic methods, GA (Genetic Algorithm) [10], [22] and PSO [11]. For CEDC-A, the accumulated slack time is used as the quantitative indicator for the profit of caching a service on an ES. PSO is same as LMPSO except that PSO does not use the lévy flight movement, and uses FF as the offloading decision maker. Similar to PSO, GA uses FF for task offloading. The numbers of populations for LMPSO, PSO and GA are all set as 100. The probabilities of mutation and crossover are set as 0.01 and 0.5, respectively. Both acceleration coefficients are set as 2.0 for PSO and LMPSO. The inertia weight is linearly decreasing with iteration time from 0.9 to 0.4.

Task scheduling methods are compared in the following three aspects.

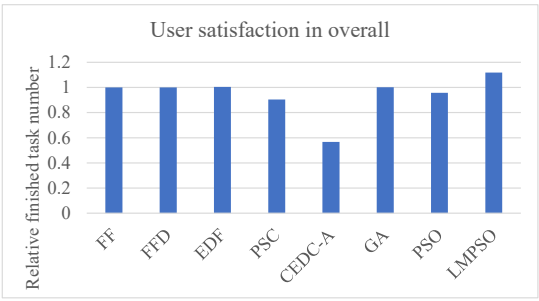
- *User satisfaction*. Three metrics are used for the quantification, the number of finished tasks (N), the accumulated computing size of finished tasks, and the amount of input data processed by finished tasks.
- *Resource Efficiency*. One of the most popular metric, resource utilization (U), is used for quantifying the resource efficiency.
- *Processing Efficiency*. The load processed per unit time. Two metrics are used, the computing size processed per second and the amount of input data processed per second.

The procedure of conducted experiment is first randomly generating 100 simulated edge-cloud systems. Then, in each generated system, values of all performance metrics are measured for all methods, and each metric value of every method is scaled by that of FF to get the relative differences of these methods. In the following, the average value from these 100 systems is reported for each metric.

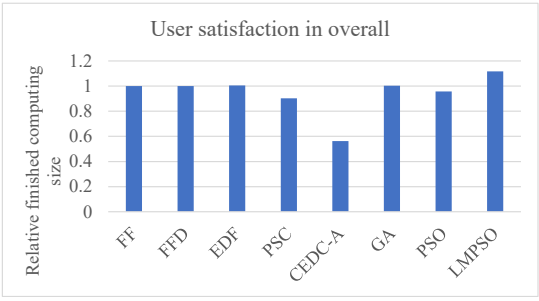
At the end of this section, several experiments are conducted for evaluating the impacts on the performance of LMPSO by some parameters of the system and the method, including the task size, the storage requirements of services, the particle number of PSO, and the coefficient of the Lévy Flight distribution.

B. USER SATISFACTION

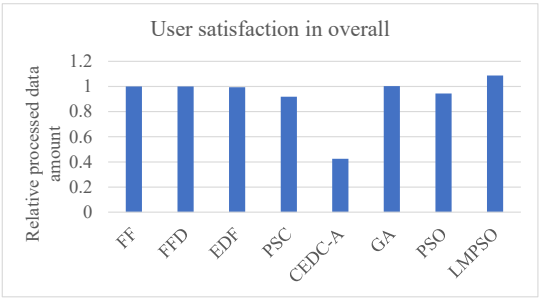
Fig. 3, 4, and 5 give the metric values in user satisfaction, in overall, in edges, and in the cloud, respectively. As shown in Fig. 3, LMPSO can complete 11.3%–97.2 more tasks, process 11.2%–98.7% more computing load and 8.34%–156% more input data, in overall, compared with other methods.



(a) Relative number of finished tasks



(b) Relative computing size of finished tasks

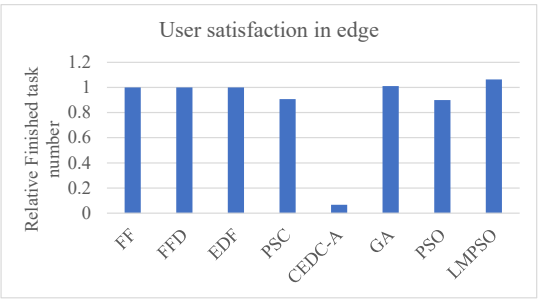


(c) Relative data amount processed by finished tasks

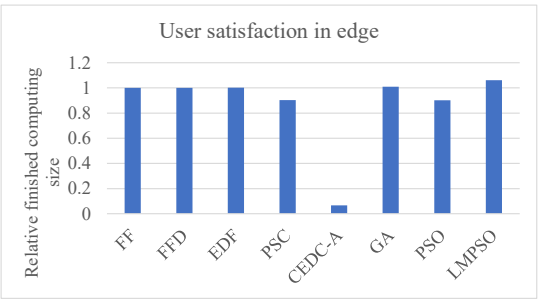
FIGURE 3. The overall user satisfactions achieved by various methods

There are mainly two improvements by LMPSO, the particle movement with lévy flight and the task offloading with three stages. To quantify the improvement of the lévy flight movement, the performance between PSO and PSO with lévy flight movement (LPSO) are compared. For evaluating the improvement of the three-stage offloading idea, each of GA, PSO and LPSO is compared with itself integrated with the three-stage offloading scheme (MGA, MPSO and LMPSO), respectively. As shown in Fig. 6, the lévy flight movement can improve 5.39% performance for PSO in the number of finished tasks. The idea of the three-stage heuristic method can improve about 11% performance in the finished task number. These two improvements guarantee that LMPSO can achieve a better user satisfaction than other methods.

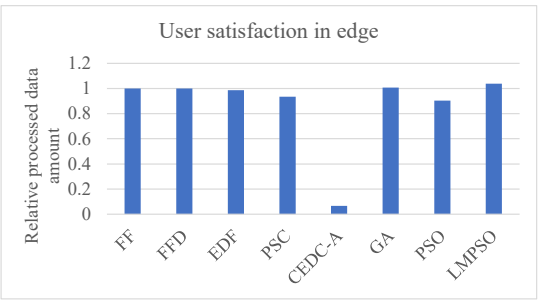
The basic idea of the three-stage task offloading method can result in a more tasks finished by the cloud. This is because the first stage try to make full use of cloud resources to finish as many tasks as possible. Therefore, in general, LMPSO has a better user satisfaction in the cloud than in



(a) Relative number of finished tasks

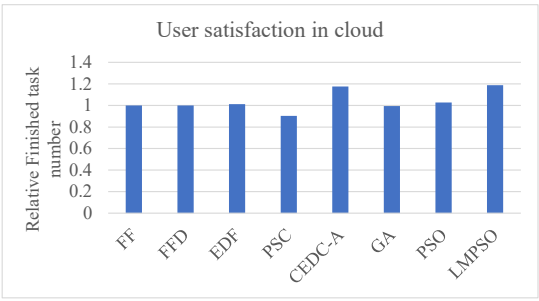


(b) Relative computing size of finished tasks

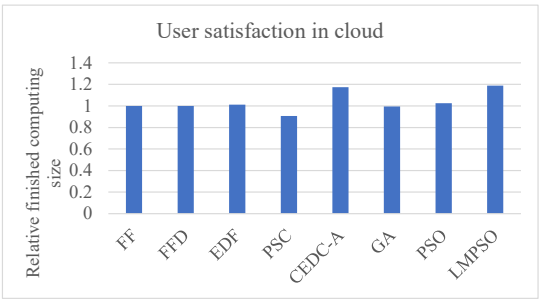


(c) Relative data amount processed by finished tasks

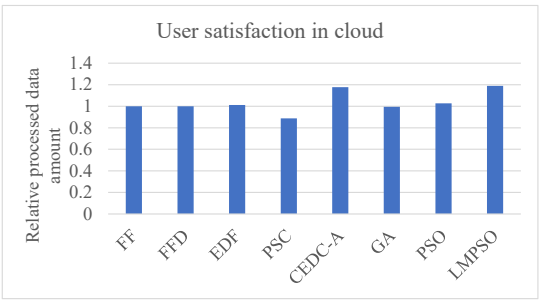
FIGURE 4. The user satisfactions achieved by various methods in edges



(a) Relative number of finished tasks



(b) Relative computing size of finished tasks



(c) Relative data amount processed by finished tasks

FIGURE 5. The user satisfactions achieved by various methods in the cloud

edges. For example, LMPSO finishes 18.8% more tasks in the cloud, as shown in Fig. 5, while only 6.34% more tasks in edges, as shown in Fig. 4, compared with FF. But there is a contrary for CEDC-A. LMPSO can finish about 15 times more tasks in edges while only 1.12% more in the cloud, compared with CEDC-A. This is mainly because CEDC-A processes too few tasks in edges, which leading to more tasks having possibility to be processed by the cloud. In addition, LMPSO re-offloads several tasks from the cloud to edges for improving the resource usage of edge resources, which is help for reducing the number of tasks offloaded to the cloud.

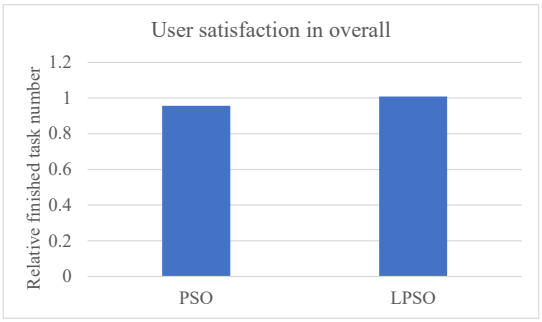
C. RESOURCE EFFICIENCY

As shown in Fig. 7, LMPSO has 1.00%–57.9% higher computing resource utilization than other methods in overall, which verifies the high resource efficiency of LMPSO. This is also benefiting from two improvement schemes. As shown in Fig. 10, the lévy flight movement improves 6.56% resource efficiency for PSO, and the idea of the three-stage offloading improves 1.00%–2.03% resource efficiency for GA, PSO and LPSO.

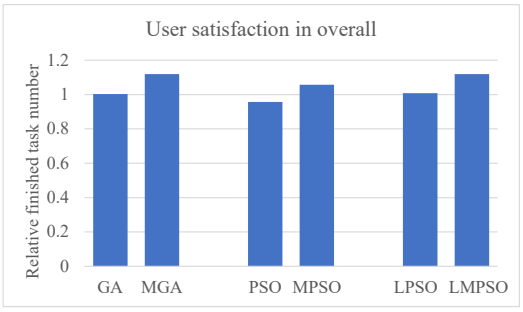
The computing resource utilization is the ratio of used computing resource amount to the amount of computing resources occupied for tasks' computing and data transfers. There are pipeline execution on each computing node that multiple tasks are offloaded to. Thus, part of data transfers are not consuming computing time, as the data transfer of one task can be concurrently executed with the computing of another task. A more tasks processed on a node lead to a greater probability of more data don't consuming computing time for their transfers, and thus result in a high resource efficiency. Thus, in general, LMPSO executes more tasks with the same edge-cloud resources, and thus achieves a better resource efficiency, compared with other methods.

D. PROCESSING EFFICIENCY

In general, the processing efficiency is decided by the speedup of parallel execution for a distributed computing system. Given a distributed system, an execution strategy providing more finished tasks imply that it achieves a greater



(a) By the lévy flight



(b) By the multi-stage idea

FIGURE 6. The improvements by the lévy flight and the multi-stage idea on the user satisfaction

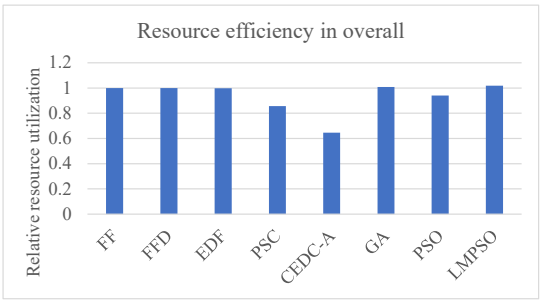


FIGURE 7. The overall resource efficiencies achieved by various methods

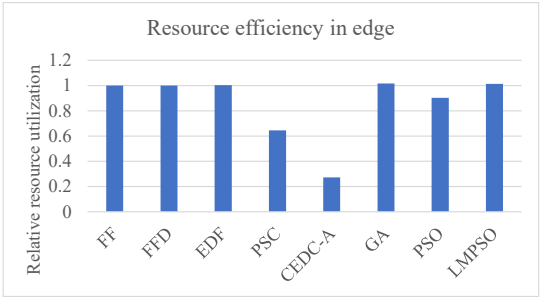


FIGURE 8. The resource efficiencies achieved by various methods in edges

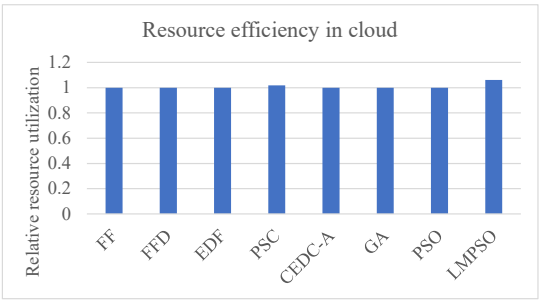
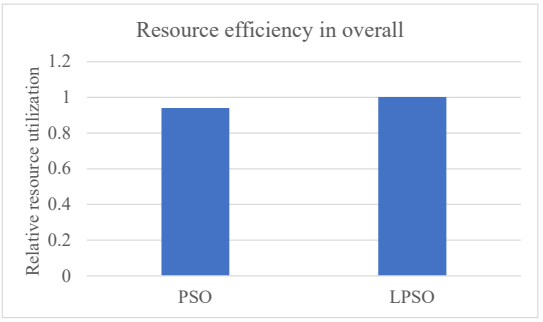
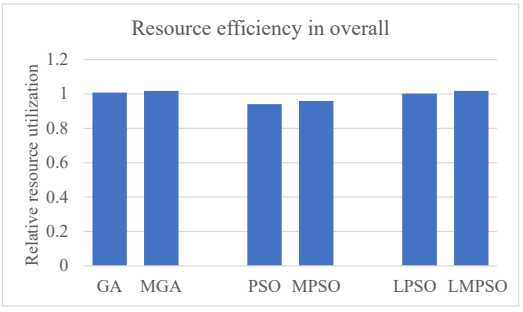


FIGURE 9. The resource efficiencies achieved by various methods in the cloud



(a) By the lévy flight

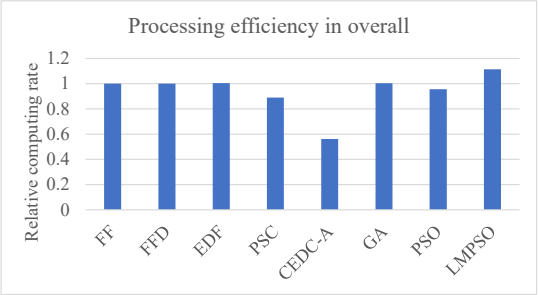


(b) By the multi-stage idea

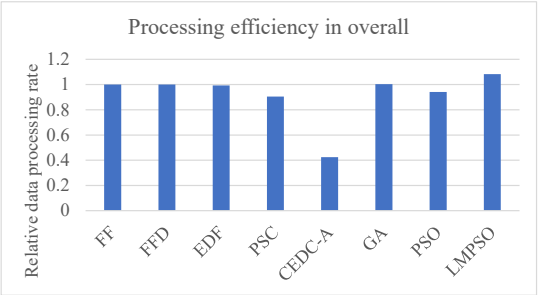
FIGURE 10. The improvements by the lévy flight and the multi-stage idea on the resource efficiency

speedup. Thus, as LMPSO finishes the most tasks, it achieves the greatest processing efficiency, compared with other methods. As shown in Fig. 11, 12 and 13, LMPSO has 10.9%–98.5% faster computing rate and 7.91%–155% faster data processing rate in overall, 3.41%–807% and 1.44%–802% in edges, 1.11%–28.3% and 1.07%–31.1% in the cloud.

In general, the processing efficiency in edges is better than that in the cloud for a task offloading method. This is mainly because the cloud has much poorer network performance than edges. This leads to a much more data transfer latency for task executions in the cloud than that in edges, and thus, more time is used for waiting data transfer for offloaded tasks to the cloud. And therefore, the processing efficiency is poor if the data transfer has long latency. Thus, both service caching and task offloading solutions largely determine the processing efficiency. This is because they influences the number of tasks offloaded to edges. Service caching solution decides which tasks can be offloaded to edges. Task offloading solution decides which tasks are offloaded to edges with limited edge resources. Therefore, this paper focuses on joint service caching and task offloading strategy, which is helpful for improving the processing efficiency, as proofed

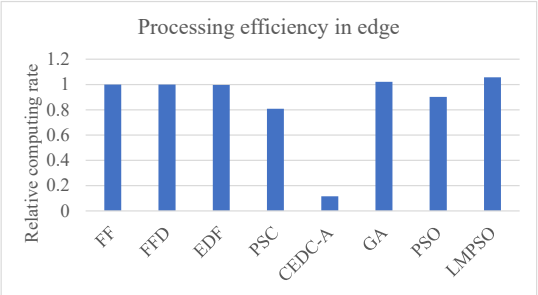


(a) Relative number of finished tasks

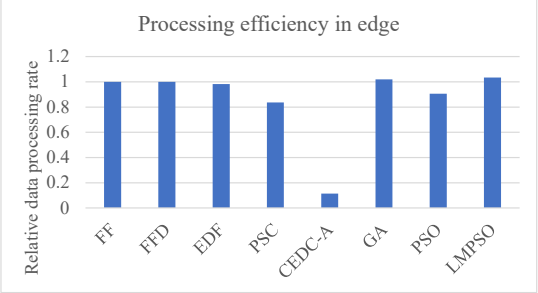


(b) Relative computing size of finished tasks

FIGURE 11. The overall processing efficiencies achieved by various methods



(a) Relative number of finished tasks

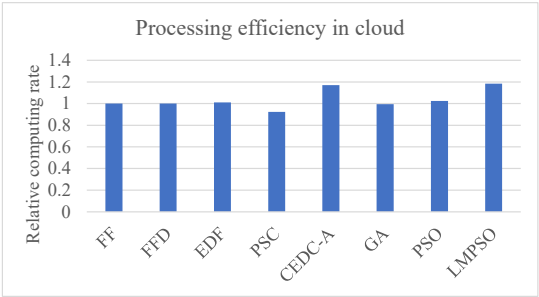


(b) Relative computing size of finished tasks

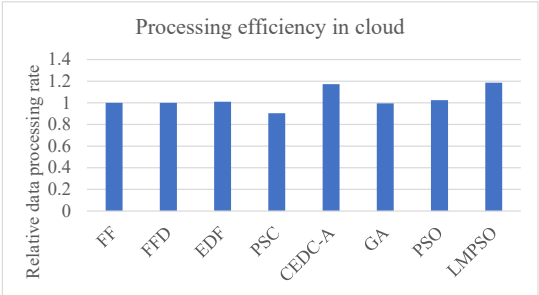
FIGURE 12. The processing efficiencies achieved by various methods in edges

by experiment results.

E. PERFORMANCE BEHAVIOUR WITH VARYING COEFFICIENTS



(a) Relative number of finished tasks



(b) Relative computing size of finished tasks

FIGURE 13. The processing efficiencies achieved by various methods in the cloud

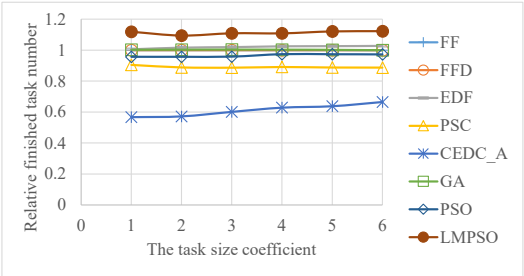


FIGURE 14. The performance changes with the size of tasks

a: Behaviour with Varying Task Size

Six groups of experiments are conducted for evaluating the performance changes of various methods with varying size of tasks. In the αth group of experiments, the computing size, the input data amount, and the deadline of each task is set in ranges of $[0.5*\alpha, 0.5*\alpha+0.7]$GHz, $[5*\alpha, 5*\alpha+1]$MB, and $[\alpha, \alpha+1]$s, where α is also called as the task size coefficient. The results are shown in Fig. 14. From the figure, it can be seen that the performance of each method is stable as the task size is increased. This proofs that our method is suitable for both small and large tasks.

b: Behaviour with Varying Service Storage Requirement

To study on the performance variation with the storage requirements of services, six groups of experiments are conducted. In the λth experiment group, the storage space required by a service is set as $[10*\lambda, 10*\lambda+10]$GB, and λ is called as the required storage coefficient. As shown in the

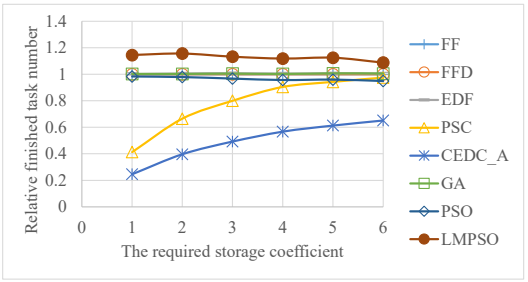


FIGURE 15. The performance changes with the storage requirements of services

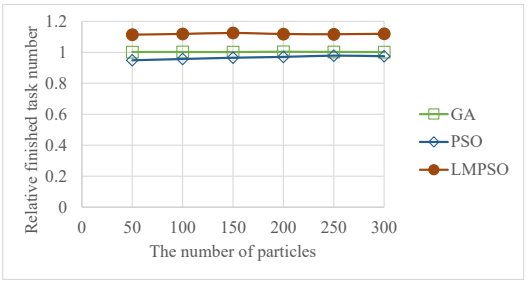


FIGURE 16. The performance changes with the number of particles (chromosomes)

experiment results, see Fig. 15, except PSC and CEDC_A, all methods have few changes with the increase of storage spaces required.

The performance of PSC and CEDC_A are increased with the storage space required by services. This is mainly because the number of services cached on ESs are decreased with their required storage space, which decreases the number of tasks offloaded to ESs, and thus reduces the room for performance improvement by caching or offloading strategy. In the extreme case, no service can be cached on ESs, and all methods have comparable performance.

c: Behaviour with Varying Individual Number

Fig. 16 gives the performance achieved by LMPSO, PSO, and GA with the increasing number of individuals from 50 to 300. As shown in the figure, the performances of LMPSO, PSO, and GA are almost no change with the number of particles or chromosomes. This suggests that one needn't take much time or effort on tuning the number of individuals, when exploiting PSO or GA based methods.

d: Behaviour with Varying Parameter of the Lévy Flight Distribution

Four groups of experiments are conducted by setting the coefficient (β) of the Lévy Flight distribution as [0, 2], [1, 3], [2, 4], and [3, 5], respectively. The results are shown in Fig. 17. From these results, it can be seen than β has almost no impact on the performance of LMPSO. This may be because the Lévy Flight distribution provides large enough steps for particle moving out of some local best positions, in

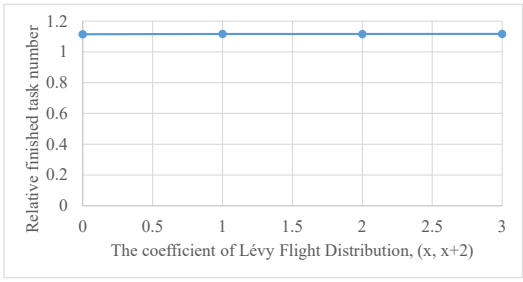


FIGURE 17. The performance changes of LMPSO with coefficient (β) of the Lévy Flight distribution. The coordinate value on the horizontal axis is x, which means $x \leq \beta \leq x + 2$.

any group of experiments. Therefore, it doesn't take effort to set the coefficient of the Lévy Flight distribution, when applying LMPSO.

V. RELATED WORK

As the increasing of user demands, the cloud computing or the edge computing alone cannot accept all of tasks requested by various mobile and IoT devices. Therefore, several published works focused on the edge-cloud computing to provide services with more efficiency by task offloading. Liu et al. [23] presented a method with the idea of earliest finish time first, which reduces the amount of the data transfer by task redundant executions, to optimize the finish time of a workflow. Sang et al. [24] proposed a heuristic task offloading method to optimize the user satisfaction with deadline constraints. The heuristic methods usually take a little time but have limited performance. Thus, some works used metaheuristic-based offloading to achieve a better performance with time-consuming at the expense of the global search ability. Zakaryia et al. [25] used generic algorithm for optimizing the overall execution time with deadline constraints. Wang et al. [11] presented a PSO-based tasks offloading method for device-edge-cloud cooperative computing to improve SLA satisfaction. Some other works exploited machine learning (ML) for task offloading. Xu et al. [26] designed a deep reinforcement learning based method to optimize the service latency for edge computing empowered Internet of vehicles. Wang et al. [27] designed a neural model to learn both offloading and time division decisions in each time slot for minimizing the delay with deadline and energy constraints. ML-based methods can provide an efficient offloading solution, but usually cost a lot of time for data training.

These above works deemed that an edge can provide all services. But in real world, there is a limited storage space for each edge, and an edge cannot support the deployment of all services. Thus, the service caching problem needed also to be studied for providing services by edge-cloud computing.

Zhao et al. [28], [29] studied on the optimization of the makespan by task offloading concerning that each ES provided only part of services due to its limited storage space. They transformed the problem to convex programming, and used progressive rounding to make offloading decision for

each task. In each ES, they exploited earliest finish time first to decide the order of task executions. Xia et al. [21] proposed an Approximation algorithm for Constrained Edge Data Caching (CEDC-A) for optimizing the overall data access delay. CEDC-A iteratively cache a data on an ES such that the caching provides the maximum profit quantified according to data access latencies. Xia et al. [30] modelled the caching problem into time-averaged optimization, and used Lyapunov to solve the optimization. These two works quantified performance by the network hop number, but in reality, the performance does not have a positive relation with the network hop number. Feng et al. [31] focused on the data caching and computing offloading problem for minimize the network cost with deadline constraints. They modelled the problem as a mixed integer nonlinear program problem, and solve its dual problem iteratively by Lagrange dual decomposition.

Different from these above related works, this work proposes to use the hybrid heuristic algorithm to solve the joint service caching and task offloading problem, to optimize the user satisfaction and the resource efficiency, for edge-cloud computing. The proposed method exploits the global search ability of PSO with the lévy flight movement, and uses a three-stage heuristic approach to improve the cooperation between edge and cloud computing.

VI. CONCLUSION

This research studies on the service caching and the task offloading problems to improve the user satisfaction and the resource efficiency for edge-cloud computing. Their joint problem is formulated as a discrete optimization problem, and a hybrid of PSO with the lévy flight movement and a three-stage heuristic method for solving the problem. Simulated experiment results verify the performance advantages of the proposed hybrid heuristic method.

This paper focuses on the static caching of services on edges, which means the caching replacement doesn't concerned during the service provisioning. When there is no storage space for caching more services on an ES, it is an option to replace a service by another new service to improve the overall performance. It is a challenging work for designing efficient caching replacement solution. If the caching replacement solution is improper, it will waste a lot of edge-cloud resources for replacing a service, leading to a much low resource efficiency. Therefore, one of the future works is to design caching replacement method with high efficiency and effectiveness.

REFERENCES

[1] M. A. Al-Shareeda, M. Anbar, S. Manickam, and I. H. Hasbullah, "Password-guessing attack-aware authentication scheme based on chinese remainder theorem for 5g-enabled vehicular networks," *Applied Sciences*, vol. 12, no. 3, p. 1383, Jan 2022.

[2] M. A. Alazzawi, H. A. H. Al-behadili, M. N. S. Almalki, A. L. Challoob, and M. A. Al-shareeda, "Id-ppa: Robust identity-based privacy-preserving authentication scheme for a vehicular ad-hoc network," in *International Conference on Advances in Cyber Security*, 2020, p. 80–94.

[3] M. A. Al-Shareeda, M. Anbar, S. Manickam, and I. H. Hasbullah, "Towards identity-based conditional privacy-preserving authentication scheme for vehicular ad hoc networks," *IEEE Access*, vol. 9, pp. 113 226–113 238, 2021.

[4] Cisco Systems, Inc., "Cisco annual internet report (2018–2023) white paper," https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html, March 2020.

[5] B. Costa, J. Bachiega, L. R. de Carvalho, and A. P. F. Araujo, "Orchestration in fog computing: A comprehensive survey," *ACM Comput. Surv.*, vol. 55, no. 2, jan 2022.

[6] A. Shakarami, H. Shakarami, M. Ghobaei-Arani, E. Nikougoftar, and M. Faraji-Mehmandar, "Resource provisioning in edge/fog computing: A comprehensive and systematic review," *Journal of Systems Architecture*, vol. 122, p. 102362, 2022.

[7] F. Saeik, M. Avgeris, D. Spatharakis, N. Santi, D. Dechouniotis, J. Violos, A. Leivadeas, N. Athanasopoulos, N. Mitton, and S. Papavassiliou, "Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions," *Computer Networks*, vol. 195, p. 108177, 2021.

[8] K. Wang, W. Chen, J. Li, Y. Yang, and L. Hanzo, "Joint task offloading and caching for massive mimo-aided multi-tier computing networks," *IEEE Transactions on Communications*, vol. 70, no. 3, pp. 1820–1833, 2022.

[9] E. H. Houssein, A. G. Gad, Y. M. Wazery, and P. N. Suganthan, "Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends," *Swarm and Evolutionary Computation*, vol. 62, p. 100841, 2021.

[10] N. Sarrafzade, R. Entezari-Maleki, and L. Sousa, "A genetic-based approach for service placement in fog computing," *The Journal of Supercomputing*, vol. 78, p. 10854–10875, 2022.

[11] B. Wang, J. Cheng, J. Cao, C. Wang, and W. Huang, "Integer particle swarm optimization based task scheduling for device-edge-cloud cooperative computing to improve sla satisfaction," *PeerJ Computer Science*, vol. 8, p. e893, 2022.

[12] M. Ghobaei-Arani and A. Shahidinejad, "A cost-efficient iot service placement approach using whale optimization algorithm in fog computing environment," *Expert Systems with Applications*, vol. 200, p. 117012, 2022.

[13] C. Liu, J. Wang, L. Zhou, and A. Rezaeipanah, "Solving the multi-objective problem of iot service placement in fog computing using cuckoo search algorithm," *Neural Processing Letters*, vol. 54, p. 1823–1854, 2022.

[14] M. Li, J. Zhang, B. Lin, and X. Chen, "Multioff: offloading support and service deployment for multiple iot applications in mobile edge computing," *The Journal of Supercomputing (In Press)*, 2022.

[15] D. Zhang, C. Gu, H. Fang, C. Ji, and X. Zhang, "Multi-strategy hybrid heuristic algorithm for single container weakly heterogeneous loading problem," *Computers & Industrial Engineering*, vol. 170, p. 108302, 2022.

[16] A. Bouaouda and Y. Sayouti, "Hybrid meta-heuristic algorithms for optimal sizing of hybrid renewable energy system: A review of the state-of-the-art," *Archives of Computational Methods in Engineering (In Press)*, 2022.

[17] X. li Lu and G. He, "Qpso algorithm based on lévy flight and its application in fuzzy portfolio," *Applied Soft Computing*, vol. 99, p. 106894, 2021.

[18] Y. Dai, D. Xu, S. Maharjan, G. Qiao, and Y. Zhang, "Artificial intelligence empowered edge computing and caching for internet of vehicles," *IEEE Wireless Communications*, vol. 26, no. 3, pp. 12–18, 2019.

[19] G. Zhang, S. Zhang, W. Zhang, Z. Shen, and L. Wang, "Joint service caching, computation offloading and resource allocation in mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 20, no. 8, pp. 5288–5300, 2021.

[20] X. Wei, J. Liu, Y. Wang, C. Tang, and Y. Hu, "Wireless edge caching based on content similarity in dynamic environments," *Journal of Systems Architecture*, vol. 115, p. 102000, 2021.

[21] X. Xia, F. Chen, J. Grundy, M. Abdelrazek, H. Jin, and Q. He, "Constrained app data caching over edge server graphs in edge computing environment," *IEEE Transactions on Services Computing (In Press)*, pp. 1–14, 2021.

[22] B. Wang, B. Lv, and Y. Song, "A hybrid genetic algorithm with integer coding for task offloading in edge-cloud cooperative computing," *IAENG International Journal of Computer Science*, vol. 49, no. 2, pp. 503–510, 2022.

[23] L. Liu, H. Tan, S. H.-C. Jiang, Z. Han, X.-Y. Li, and H. Huang, "Dependent task placement and scheduling with function configuration in edge com-

puting," in *2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*, 2019, pp. 1–10.

[24] Y. Sang, J. Cheng, B. Wang, and M. Chen, "A three-stage heuristic task scheduling for optimizing the service level agreement satisfaction in device-edge-cloud cooperative computing," *PeerJ Computer Science*, vol. 8, p. e851, 2022.

[25] S. A. Zakaryia, S. A. Ahmed, and M. K. Hussein, "Evolutionary offloading in an edge environment," *Egyptian Informatics Journal*, vol. 22, no. 3, pp. 257–267, 2021.

[26] X. Xu, Z. Fang, L. Qi, W. Dou, Q. He, and Y. Duan, "A deep reinforcement learning-based distributed service off loading method for edge computing empowered internet of vehicles (in chinese)," *Chinese Journal of Computers*, vol. 44, no. 12, pp. 2382–2405, 1 2021.

[27] X. Wang, Z. Ning, L. Guo, S. Guo, X. Gao, and G. Wang, "Online learning for distributed computation offloading in wireless powered mobile edge computing networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 8, pp. 1841–1855, 2022.

[28] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 1997–2006.

[29] ——, "Offloading tasks with dependency and service caching in mobile edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 11, pp. 2777–2792, 2021.

[30] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 281–294, 2021.

[31] H. Feng, S. Guo, L. Yang, and Y. Yang, "Collaborative data caching and computation offloading for multi-service mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 9, pp. 9408–9422, 2021.