A Pareto-Efficient Task-Allocation Framework Based on Deep Reinforcement Learning Algorithm in MEC

Wenwen Liu, Sinong Zhao, Zhaoyang Yu, Gang Wang(✉), and Xiaoguang Liu(✉)

College of Computer Science, TJ Key Lab of NDST, Nankai University, Tianjin, China
{liuww,zhaosn,yuzz,wgzwp,liuxg}@nbjl.nankai.edu.cn

Abstract. Mobile-edge computing (MEC) has emerged as a promising paradigm that moves tasks running in the cloud to edge servers. In MEC systems, there are various individual requirements, such as less user-perceived time and lower energy consumption. In this case, substantial efforts have been paid to task allocation, aiming at enabling lower latency and higher resource utilization. However existing studies on multiple-objectives task allocation algorithms rarely consider the Pareto efficient problem, where no objective could be further improved without vitiating the other objectives optimization. In this paper, we propose a Pareto-efficient task-allocation framework based on a deep reinforcement learning algorithm. We give the formal formulations for objectives and construct a multi-objectives' optimization model for task allocation. Then a Pareto efficient algorithm is proposed to solve the problem of conflicting among multi-objectives. By coordinating multi-objectives parameters get from Pareto efficient algorithm, the deep reinforcement learning model takes a Pareto-efficient task allocation to improve real-time and resource utilization performance. We evaluate the proposed framework over various real-world tasks and compare it with existing allocating tasks models in edge computing networks. By using the proposed framework, we can get an accuracy that not be lower than 90% under the 0.6 s latency requirement. The simulation results also show that the proposed framework achieves lower latency and higher resource utilization compared to other task allocation methods.

Keywords: Mobile edge computing · Pareto-efficient · Task-allocation framework · Deep reinforcement learning algorithm

This research is supported in part by the National Science Foundation of China under Grant 62141412 and Grant 61872201, in part by the Science and Technology Development Plan of Tianjin under Grant 20JCZDJC00610, in part by the Key Research and Development Program of Guangdong under Grant 2021B0101310002, and in part by the Fundamental Research Funds for the Central Universities.

H. Gao et al. (Eds.): CollaborateCom 2022, LNICST 461, pp. 197–213, 2022.
https://doi.org/10.1007/978-3-031-24386-8_11

1 Introduction

With the development of Internet of Things (IoT) and Artificial Intelligence (AI), the number of terminal tasks has increased violently. This caused the increasing energy consumption and latency. Mobile Edge Computing system (MEC) [2, 13,14] extends the tasks running in the cloud into edge servers to guarantee QoS requirements. It integrates computing resources, storage space and network bandwidth to ensure system lower latency [6,22], higher resource efficiency [30], and better security [23,26,28]. This new kind of computing paradigm encounters some new challenges. For example, it is an urgent issue that how to guarantee the normal operation of tasks under limited resource capacity and response time.

Some research has addressed the above issue and put forward effective task allocation solutions. Zhang et al. [27] designed a Load-Aware Resource Allocation and Task Scheduling (LA-RATS) algorithm to deal with both delay-tolerant and delay-sensitive mobile applications. They first formulated a resource allocation model for delay-sensitive and delay-tolerant requirements, and then proposed a task back filling mechanism that has two merits: (1) by using a backward shifting strategy, it could full use of the idle resource, (2) by avoiding unnecessary queue growth for VMs to save energy consumption and running time. Wang et al. [20] proposed a unified Mobile-Edge Computing and Wireless Power Transfer model (MEC-WPT model) that addressed the latency-limited practical scenario. It improves the MEC performance by jointly optimizing (1) the energy transmit beamforming at the AP, (2) the CPU frequencies, (3) the offloaded bit numbers, and (4) the time allocation among users. Based on the above work, they developed an optimal resource allocation scheme that minimizes the total energy consumption under the constraint that users' individual computation latency. Jiao et al. [21] proposed an online resource optimization algorithm that points at a gap-preserving transformation of the problem. It offers a feasible solution with a designed logarithmic objective for edge cloud resource allocation over time.

However, the current task-allocation algorithms rarely consider the Pareto efficient problem among multiple-objectives. The principal reason is that the conflicts among different objectives, and it's difficult for us to optimize multi-objectives at the same time. In an edge computing system, the CPU utilization rate and task running time are not entirely consistent.

To address the issues aforementioned, in this paper we study Pareto-efficient task allocation and present a **P**areto-efficient task allocation framework for improving resource **U**tilization and **R**eal-timing in **E**dge computing networks, we call it **Pure**. The contributions of this paper are summarized as follows:

- First, we formulate the objectives (response time and CPU energy consumption) and construct a multi-objectives optimization model for task allocation in edge computing networks.
- Second, we propose a Pareto efficient algorithm to solve the problem of conflict among multi-objectives. It uses the scalarization method to transform the multi-objectives problem into a single objective problem and gets QoS optimization.

– Third, by coordinating multi-objectives parameters get from Pareto efficient algorithm, the deep reinforcement learning model takes a Pareto-efficient task allocation to improve real-time and resource utilization performance.
– Finally, we evaluate Pure over various real-world tasks and compare it with three existing popular models in a simulation environment. The simulation results show that Pure's better performance in terms of latency, accuracy and resource utilization.

The rest of the paper is organized as follows. Section 2 provides related work and Sect. 3 describes the detailed design of Pure that includes Pure's architecture overview, quantifying the performance objectives and construction of the multi-objectives optimization model, the description of the Pareto efficient algorithm and the DRL-based Pareto-efficient task allocation model. The evaluations and analysis are displayed in Sect. 4. Section 5 concludes this paper.

2 Related Work

Task allocation is one of the main challenges in MEC, and a variety of research interests are emerging. In this section, we mainly focus on two aspects of multi-objectives task allocation: the content of multiple objectives and how to solve the multi-objectives model.

In the prior work, different optimization objectives and solution methods are considered in task allocation. In [19], the authors provided a multi-objectives task allocation mechanism in a multi-robot system. They mainly considered the tasks' actual energy and completion time objectives. Based on these, they built the multi-robot dynamic task allocation problem and used a genetic algorithm to solve the problem. Although the proposed algorithm is so flexibility that it can be implemented in other domains, its computational complexity is very high, and the tasks allocation could require high computational resources. Dinh et al. [4] proposed an optimization framework of offloading from a single mobile server to multiple edge servers. They built a multi-objectives model to minimize the execution latency and the edge servers' energy consumption objectives. A linea relaxation-based approach and a semidefinite relaxation (SDR)-based approach were proposed and be used to achieve the near optimal performance. The evaluation results demonstrated the framework's performance improvement in terms of energy consumption and execution latency when multiple edge servers and CPU frequency are considered. However, this paper was not considered about the Pareto-efficient of multi-objectives model. In [17], Ziwen Sun et al. proposed an Attack Localization Task Allocation (ALTA) algorithm in edge sensor networks. They mainly focused on the total task execution time, total energy consumption and load balance objectives. In the algorithm, the multi-objectives binary particle swarm optimization method is used to determine the nodes joining to locate attacks in order to prolong the lifetime of networks during locating attacks' position. Same as before, this paper was not considered about the Pareto-efficient of multi-objectives model.

In particular, some work paid more attention to resource scheduling in edge task allocation. In [16], Yan Sun et al. proposed a two-level resource scheduling model to achieve a resource scheduling scheme. The scheme constructs multi-objectives problems that optimizing the service latency and the overall stability of task execution. In order to solve multi-objectives optimization problems, they proposed a novel resource scheduling scheme using an improved Non-dominated Sorting Genetic Algorithm II (NSGA-II). Although the scheme could reduce the service latency and improve the stability, it did not consider how to reduce the cost of resource. In [12], Liqing Liu et al. proposed a multi-objectives optimization problem that minimizes the energy consumption, execution delay, and payment cost. They used the scalarization method to transform the multi-objectives optimization problem into a single-objective optimization problem. And Interior Point Method (IPM) was applied to solve the a single-objective optimization problem. The simulation results showed that the proposed model could reduce the accumulated error and improve the calculation accuracy effectively. But it also could not consider the Pareto-efficient problem of multi-objectives model. Chu-ge Wu et al. proposed a fuzzy logical offloading strategy based on multi-objectives resource allocation in edge computing [12]. The optimization objectives is both agreement index and robustness. A multi-objective Estimation of Distribution Algorithm (EDA) was designed to solve and optimize the fuzzy offloading strategy. Similarly, it was also not consider the Pareto-efficient problem of multi-objective model.

To sum up, it can be inferred that the energy consumption and latency are important factors that need to be optimized in edge scenarios. Most of the above references have optimized these two factors. But these previous works were rarely considered the Pareto-efficient problem of multi-objectives model. However the Pareto-efficient problem can be further improve the efficiency between delay and resource utilization. In this paper, we present the quantitative formulas of response time and CPU energy consumption, and construct a multi-objectives optimization model for task allocation. Further, we achieve the parameters for transforming the multi-objectives problem into a single-objective problem by Pareto efficient algorithm and get the efficient tasks' collocation by the DRL algorithm. The details are as follows.

3 The Pure Framework

3.1 The Architecture Overview

Generally, a MEC system involves three tiers: 1) the terminal device tier contains end users, mobile devices and sensors, 2) the edge server tier contains multiple interconnected edge servers, such as cloudlets, and 3) the cloud tier acts as a monitor of the whole system. Here we mainly pay attention to the edge server tier. Pure aims to take task allocation to improve edge server CPU utilization and reduce the task execution time. There are three steps of Pure:

– **Quantifying the performance objectives and constructing the multi-objectives optimization model.** We quantify the performance objectives

that affects QoS requirement, including the response time and the energy consumption of CPU of each task. Based on this, we construct a multi-objectives optimization model for task allocation.

- **The Pareto efficient algorithm.** This step we use scalarization technique to achieve the parameters for transforming the multi-objectives problem into a single-objective problem. This also makes Pareto-efficiency be guaranteed, where no objective can be further optimized without weakening other objectives.
- **The deep reinforcement learning-based Pareto-efficient task allocation model.** Based on the single-objective model gets from the Pareto efficient algorithm, in this step we introduce how to adapt deep reinforcement learning technology completing the Pareto-efficient task allocation model.

We will detail each step in the following subsections.

3.2 Quantifying the Performance Objectives and Constructing the Multi-objectives Optimization Model

Interdependent Tasks Model. This part, we mainly introduce the interdependent tasks model.

Symbols representation of corresponding variables are explained in Table 1. Given a task set $N = \{1, 2, ..., n\}$, an edge server set $M = \{1, 2, ..., m\}$. For an edge server $j(1 \leq j \leq m)$, its dominant frequency is f_j. For an task $i(1 \leq i \leq n)$, the amount of data to be input during execution is d_i^U, the total number of CPU clock cycles required during execution is c_i, the average upload rate from task i to edge server j is v_{ij}^U, the average download rate is v_{ij}^D. And there is an allocation indication variable set $L = l_{i,j}(1 \leq i \leq n, 1 \leq j \leq m)$, i.e.,

$$l_{i,j} = \begin{cases} 1, & \text{if task } i \text{ is allocated to edge server } j, \\ 0, & \text{otherwise.} \end{cases}$$

The Execution Time Model. In this part, we mainly focus on the execution time. We can get the execution time of task i from following equation:

$$t_i = \sum_{j=1}^{m} l_{ij} * (c_i / f_i). \tag{1}$$

And the whole application' (including n tasks) execution time can be calculated as following equation:

$$T = \sum_{i=1}^{n} (\sum_{j=1}^{m} l_{ij} * (c_i / f_i)). \tag{2}$$

The Computational Energy Consumption Model. Considering that CPU is the most energy consuming factor in edge servers, we take the CPU utilization

Table 1. Table of notations and descriptions.

Notation	Description
N	The set of the tasks
M	The set of the edge servers
f_j	The dominant frequency of edge server j
d_i^U	The amount of data to be input during execution task i
c_i	The total number of CPU clock cycles required during execution task i
v_{ij}^U	The average upload rate from task i to edge server j
v_{ij}^D	The average download rate from task i to edge server j
L	The allocation indication variable set
$l_{i,j}$	The allocation indication variable when task i is allocated into edge server j $(1 \leq i \leq n, 1 \leq j \leq m)$

as the computational energy consumption model in system. The dynamic power calculation model of CPU in edge server j is given in [3,25], that is

$$P_j(s) = \sigma + \mu s^{\alpha}, \tag{3}$$

where σ is the static power, μ and α are constants that relate to the specific hardware device, and $\alpha \geq 1$, s is the running speed of edge server j, which is proportional to the frequency.

Thus the energy consumption of task i running in edge server j is

$$P_i = \sum_{j=1}^{m} l_{ij} * P_j(s). \tag{4}$$

And the whole application' (including n tasks) energy consumption is

$$P = \sum_{i=1}^{n} (\sum_{j=1}^{m} l_{ij} * P_j(s)). \tag{5}$$

Multi-objectives Optimization Model. To sum up, this paper constructs the following multi-objectives function,

$$\min \Gamma(T, P) \tag{6}$$

$$\Gamma(T, P) = \gamma T + (1 - \gamma) P \tag{7}$$

$$\text{s.t.} \quad \gamma \in [0, 1], \tag{8}$$

$$T \leq \epsilon, \tag{9}$$

$$P_i \in [P_{min}, P_{max}], \tag{10}$$

$$\sum_{j \in m} l_{ij} = 1. \tag{11}$$

where ϵ denotes the execution time threshold. P_{min}, P_{max} represent the minimum and maximum energy consumption that the current edge server can load.

Objective functions (6) and (7) represent to minimize the application's execution time and computational energy consumption. We use a customized weighted product method [18] to approximate Pareto solutions, with multi-objectives fusion and optimization goal. Constraint (8) limits the range of γ, and γ is the balance factor of T and P. Constraint (9) guarantees the whole application's execution time is not to exceed the time threshold ϵ. Constraint (10) keeps the energy consumption of task i cannot exceed the upper and lower limits of the edge server's capacity. Constraint (11) indicates that each task is allocated and can only be allocated to a unique edge server.

In the above multi-objectives optimization model, the goal of task allocation is to map n tasks to m edge servers, so as to minimize the energy consumption on the premise of reducing the task execution time as much as possible. In this way, given a set of tasks and edge servers, under the execution time and server energy consumption thresholds, the multi-objectives optimization problem is transformed into an optimal task allocation problem to minimize execution time and energy consumption. Therefore, how to get a solution that optimizes for two objectives in the sense that no objective can be further improved without vitiating the other objectives optimization is introduced as the following subsection.

3.3 The Pareto Efficient Algorithm

Currently, the existing solutions for Pareto optimization mainly includes two categories: the heuristic search method and scalarization method [11]. In order to obtain the Pareto efficient solution of the multi-objectives model in Subsect. 3.2, we use scalarization method to transform multi-objectives problem into single objective problem, and then solve the value of γ.

There we assume that there are δ optimization objectives in the multi-objectives optimization model. $\Gamma(\theta)$ represents the model to optimize all objectives (corresponding the objective function (6)). Suppose δ objectives have δ differentiable loss functions $l_i(i \in (1, 2, ..., \delta))$ correspondingly, then the loss function $L(\theta)$ of δ objectives is:

$$L(\theta) = \sum_{i=1}^{\delta} \gamma_i l_i. \tag{12}$$

Based on above, we optimize δ objectives that is equal to minimizing $L(\theta)$. We use scalar technology to combine multiple objectives into one objective, and then solve it, get the value of γ. The problem is transformed into solving the minimization loss function $L(\theta)$ process:

$$\min \quad || \sum_{i=1}^{\delta} \gamma_i \nabla_\theta l_i ||_2^2 \tag{13}$$

$$\text{s.t.} \quad \sum_{i=1}^{\delta} \gamma_i = 1, \gamma_i \geq c_i, \forall i \in 1, 2, ..., \delta. \tag{14}$$

This is a non-negative least squares problem, paper [11] gave the whole solution to this problem. Due to page limited, we omit the details of the solution process. From the result of the solution, we get the generated Pareto Frontier set, the solution with minimum requirement to get the γ value. Finally, we get the single objective model depending on the choice of fairness. Solving it $\gamma = 0.73$. Unless explicitly stated we use $\gamma = 0.73$ in our experiments. Next subsection will describe the DRL-based Pareto-efficient task allocation model.

3.4 The DRL-Based Pareto-Efficient Task Allocation Model

Deep Reinforcement Learning (DRL) [9,10] is one of the machine learning techniques that combines the perception ability of deep learning with the decision-making ability of reinforcement learning. It drives agent(s) learning to maximize reward while interacting with an uncertain and varying environment in deep learning networks. We choose it to take tasks allocation for three reasons. Firstly, DRL is a Markov decision process, in which future strategies are only related to the present state, not past state. Secondly, the well-trained DRL model is running fast, while the edge task allocation also requires real-time characteristic. Thirdly, the DRL has the advantages of scalability and versatility. Our well-trained DRL model can adapt to current scenario through corresponding training dataset, and we can cope with the change of various scenarios by changing corresponding training dataset.

In order to deal with the task allocation in the case of edge servers, we apply Deep Q-Learning Network (DQN) [5].

A Markov Decision Progress (MDP) of our problem is defined as a five-tuple $< S, A, P(s,a), R(s,a), s_0 >$, S is a finite set of states appeared in the environment, A is a finite set of actions, $P(s,a)$ is a next state transition probability matrix gets from action a in the state s, $R(s,a)$ is the reward function that indicates how well the agent is doing after taking the action a in the state s, and s_0 is the initial state in the environment.

Next, we describe the adaptive DRL task allocation approach including state space, action space, reward function, offline training and online allocation.

State Space: Given the state set $S : \{s_0, s_1, ..., s_t, ..., s_\omega\}$, s_0 is the initial state, ω denotes the states number. For the current task i allocation, we want to deal with how to minimize the energy consumption while ensuring less execution time according to the definition of function (6–7), constraints (8–11)). We adopt the initial state s_0 is the state that all servers is unoccupied and no tasks has been assigned, and $s_t = L_t$, L_t is an $n * m$ matrix representing n tasks' allocation scheme in m edge servers at state t. If task $i(i \in n)$ is allocated to edge server $j(j \in m)$, $l_{i,j}$ is 1, otherwise it is 0. So L is a (0,1) allocation matrix that makes up of $l_{i,j}$.

Action Space: At each state, assuming that n tasks come to our system to be allocated, there are m edge processors. Hence, each task has m optional processing positions to be allocated, the action space size is n^m. This leads to unbearable amount of computation and training time on edge processors. In our model, for each task i to be allocated, we keep the agent to take only one action (0 or 1, 0 denotes task i is not allocated into the current edge server, 1 denotes i is allocated into the current edge server). Therefore the task i action space is $\{a_1, a_2, ..., a_m\}$, and n tasks action space size is $n * m$.

Reward Function: The reward function $R(s, a)$ indicates how well the agent is doing after taking the action a in the states s, it helps the learner learn the feedback value of action and impacts the network learning quality highly.

Our goal is to get tasks allocation through solving the multi-objectives optimization model in Subsect. 3.3. Base on the execution time and energy consumption optimization objectives, we define a reward function about them. We denote T_i and P_i as the sum of execution time and energy consumption of entire tasks from state $i-1$ to state i. Then we normalize them to get the reward value in state i which has a regular form for DRL algorithms.

$$r_i = \gamma * \frac{T_{i-1} - T_i}{T_0} + (1 - \gamma) * \frac{P_{i-1} - P_i}{P_0}. \tag{15}$$

In Eq. (15), we first get the execution time T_0 and energy consumption P_0 in initial state s_0, and take them as baseline. To have a comparison with the last state s_{i-1}, we do the corresponding values in state s_{i-1} minus the values in state s_i, then divide the result by s_0's corresponding value. This reward function returns the reward value of the state i. Obviously, a higher value of r_i stands for the greater reward, the more execution time and energy consumption the current state save compared to the prior state s_i, and the more effective the current action is.

Offline Training. Based on the above, the well-trained model could be generated offline. Algorithm 1 illustrates the training process. In Algorithm 1, we set x epochs(that makes the training process converge enough), and each epoch of training has y steps. At the beginning of the algorithm, we require the tasks queue Q_{task}, the execution time constraint ϵ and the limit of energy consumption $[P_{min}, P_{max}]$ as the input factors. Then the s_0 and empty task allocation matrix L are initialized. We run the tasks in FIFO order from Q_{task}, and periodically sample the edge servers' resource conditions. Then allocate the task according to the actions and compute the rewards after the actions are performed. At last update the parameters of the neural network. Repeat the step and epoch until convergence or getting the maximum epoch threshold. The DRL model is well-trained.

Algorithm 1. DRL Training

Require:
Build the DRL neural network architecture A;
Put tasks into priority queue Q_{task} according to FIFO;
$[P_{min}, P_{max}] \leftarrow$ the limit of energy consumption;
$\epsilon \leftarrow$ the execution time constraint;
Ensure: The well-trained DRL model.
Initialization:
Replay memory B to capacity C_B;
$s_0 \leftarrow$ initial state;
$L \leftarrow$ the task allocation matrix;
for *epoch* $\leftarrow 1$ to x **do**
  Get the current task from Q_{task};
  Update the current state s;
  Update the task allocation matrix L;
  for *step* $\leftarrow 1$ to y **do**
    if $e < \epsilon$ **then**
      Select the action a by running A;
      Get the next state s_{step+1} and compute the next edge reward r;
      Store transition(s, a, r, s_{step+1}) in B;
      Update the network by sampling transitions;
      $s = s_{step+1}$.
    end if
  end for
end for

After all the epochs are performed or the model gets convergence, the reinforcement learning neural network is adjusted to the best state. And the models in different epochs are saved. We can obtain a well-trained model which has been loaded and use it inference without retraining. The algorithm time complexity is $\mathcal{O}(x * y)$.

To avoid the problem of hard convergence in training, we get experience replay method to alleviate correlation in the sample sequence (the detailed operation is not described here, refer to [24]).

Online Allocation. After the training process, we obtain a well-trained model, with the network parameters corresponding to each action. However, what we need is the current task's optimal allocation of the inference system. When a new task starts coming into task queuing by FIFO, the trained model periodically observes the system states and takes corresponding actions online. Repeat the process until all the epochs are performed or the model gets convergence, then return the final task allocation matrix L.

4 Evaluations

In this section, we evaluate the performance of Pure with simulation experiments in edge computing scenarios.

4.1 Experiments Setup

We use the cloudlet-discovery project [15] to establish our edge computing simulation environment, and 2-tier architecture was simulated, which includes edge tier and cloud tier. VMware 12 tool is used to install multiple virtual machines to mimic edge servers. We simulate three edge servers in edge tier and one cloud server in cloud tier, their configuration is listed in Table 2, and they are all inter-connected via WiFi. The Network Time Protocol (NTP) tool is applied to synchronize time among edge servers and cloud server. The *perf* tool is used to monitor the hardware performance in Linux. The bandwidth is set to 8Mbps according to Internet connection speeds in different countries in [1].

In order to catch the CPU features, we use tested tasks are divided from autonomous driving applications. We choose Image classification, Real-time positioning, Feature extraction and Object detection application domains under different models and datasets to enrich the number of tasks, such as image classification applications under Resnet-50, Yolo, MobileNet-v3 (etc.) models used Imagenet, Cifar-100, and KITTI [7] (etc.) datasets. And eventually we get 216 tasks in different types.

Baselines. We employ the following task allocation methods as baselines.

- **The shortest distance task allocation strategy (SD)** [15]: It selects the closest edge server to requesting terminal device. Although the shortest distance method reduces the network transmission distance, it does not consider whether the shortest distance edge server is suitable for task execution or not. Therefore, it may lead to multiple allocations, so as to prolong the delay.
- **The prioritized task scheduling strategy (PS)** [8]: It prioritizes the tasks according to a priority policy. Generally, the task with tighter time limits should have higher priority to be allocated. The latest allowable start time (LAST) of task is defined as priority score. That is the task with earlier LAST should have higher priority, and be allocated to edge server preferentially.
- **Greedy algorithm (GA)** [29]: For each task, GA takes out the maximum resource capacity by greedy algorithm, then places them into the corresponding edge server under the QoS requirements, at last updates allocated servers tasks combination and resource capacity.

Table 2. Hardware configurations

Equipment	CPU frequency	No. core	Memory
Laptop	3.6 GHz	4 cores	8 GB
Edge server A	2.4 GHz	2 cores	4 GB
Edge server B	2.4 GHz	4 cores	8 GB
Edge server C	1.2 GHz	2 cores	1 GB

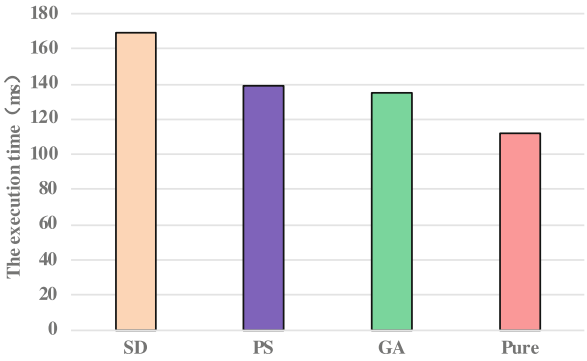

Fig. 1. The comparison of four task allocation strategies' execution time

4.2 The Execution Time Evaluation

Latency is an important metric in edge computing scenarios. Here we mainly focus on the execution time in formula (2). Figure 1 shows the execution time of different task allocation strategies. It can be seen that when performing tasks, Pure achieves the lowest execution time and the SD task allocation strategy's execution time is rather high. The execution time of PS and GA are lower than that SD and higher than Pure.

The reason for this is that SD only considers the distance between mobile device and edge server, it does not consider whether the closest server meets the network congestion and insufficient computing resources, re-allocation is possible. Compared with SD, PS takes consideration of the tasks' running time and the available resources of servers, so it ensures the tasks with stricter time constraints are allocated with more sufficient computing resources, thereby reducing overall execution time. GA gets greedy algorithm to select the current most suitable edge server under the execution time threshold. Compared with above three baselines, the Pure leverages the Pareto efficient algorithm and well-trained DRL allocation algorithm to take allocation. Online selection edge server time is short. Therefore, Pure obtains the minimum execution time.

4.3 The Reliability in Mobile Scenario Evaluation

To better understand of the reliability in mobile scenario, we explore the accuracy ratio of image recognition tasks, that is, the probability of correct image recognition ratio. Figure 2 plots the accuracy ratio of four strategies under the 0.6s latency requirement while meeting the mobility demand. The reliability is measured by accuracy under three states of the mobile terminals including static, low mobility and high mobility. The speed of mobile terminal at low mobility is 10 miles per hour (MPH) and the speed at high mobility is 35 MPH. The corresponding observations and analysis are as follows.

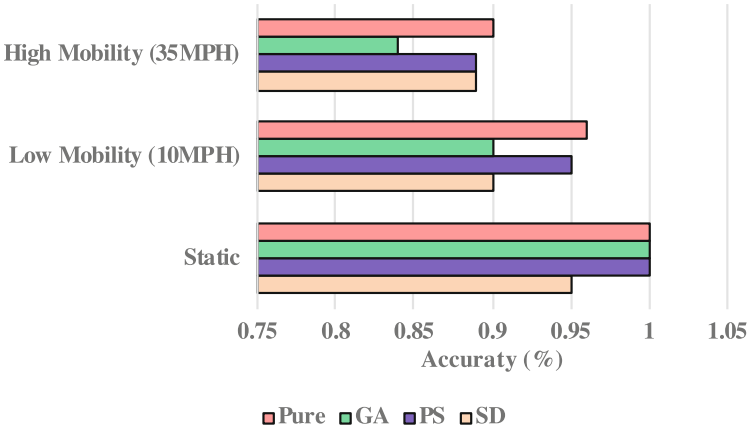

Fig. 2. The comparison of reliability of four task allocation strategies in mobile scenario

First, the accuracy ratios decrease as the speeds of the mobile device increase. The reason is obvious. On the one hand, the higher speed of mobile terminal makes the wireless network unstable. On the other hand, the quality of the network link between different cellular unit may decrease in the mobile environment.

Second, for four strategies, we can see that the reliability of Pure is higher than the others at various moving speed. Compared with SD, the Pure allocates the edge server with sufficient computing resources while ensuring the execution time in the mobile process, however the SD may have the problem of insufficient computing resources and network congestion, therefore the Pure's result is higher than SD. On the other hand, for the PS and GA, no matter how fast the mobile terminal moves, it will select the edge server in the current link range, so their accuracy rates will not be affected largely. And Pure will select the edge server of network transmission quality in the process of moving. Furthermore, at the latency requirement of 0.6 s, the reliability of Pure is able to achieve not be lower than 90%, which maintains the highest reliability at different moving speeds.

4.4 The CPU Cost of Edge Servers

In this subsection, we monitor the CPU utilization of the four strategies in the dynamic time situation. There we run the 10 tasks to record the changes in CPU utilization of SD, PS, GA and Pure strategies, which is shown in Fig. 3. Analysis for each algorithm is as follows.

For SD, from the start time of the task (CPU utilization up to non-zero value, we can get this from abscissa) in Fig. 3 (a), (b), (c), the first task is allocated to edge server A, then to edge server B and finally to edge server C. According to the duration running time of edge servers, there are more tasks performed on edge servers A and B compared with edge server C. In addition, because resource

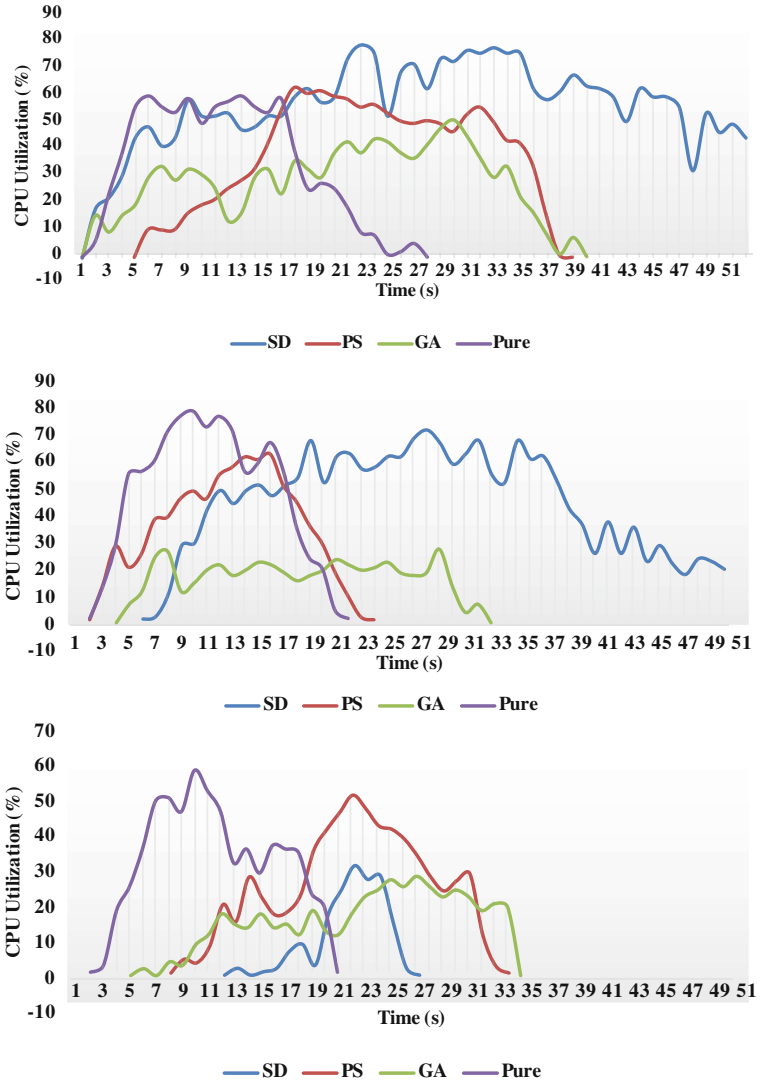

Fig. 3. The CPU utilization of (a) Edge server A (b) Edge server B and (c) Edge server C.

optimization is not considered, the total CPU resource loads (the shadow part covered by the blue curve in three figures) on the servers are very high.

For PS, we get from PS's curve that it first allocates task to edge server B based on the time constraints and available resources on the edge server. Since the resources of edge server B are occupied by the first task, the other tasks are allocated to edge servers A and C successively, which have more available computing resources. Judging from the duration running time, the CPU load

is more balanced than SD and GA, and the time to finish all the tasks is also shorter them.

For GA, it uses the greedy algorithm to assign tasks to the server A, server B, and server C in time order. The tasks' execution time of GA is longer than PS and Pure, because GA is not taking much more consideration on execution time, the same is true for imbalance CPU utilization.

Last but not least, for Pure, three servers start executing tasks almost simultaneously and the completion time is about 39 s (we get the longest running time in three servers). In addition, the total CPU load (the shadow area covered by purple curves) is much smaller and more balanced than other baselines. It shows that the proposed algorithm Pure can balance decreasing execution time and CPU utilization well.

5 Conclusions

In this paper, we present Pure framework to ensure that tasks with various demands run efficiently in MEC system. It constructs a multi-objectives optimization model for task allocation, and a Pareto efficient algorithm is proposed to solve the problem of conflict among multi-objectives. Then a deep reinforcement learning model takes a Pareto-efficient task allocation for improving real-time and resource utilization performance. The evaluations were presented to illustrate the effectiveness of the Pure framework and demonstrate the superior performance over the existing traditional task-allocation strategies. Specifically, Pure could get much lower latency, more accurate result and lower CPU cost.

In the future, some other issues could be considered. We will try to take Pure test more types resources, such as I/O, bandwidth and so on.

References

1. List of countries by internet connection speeds (2020). https://en.wikipedia.org/wiki/List_of_countries_by_Internet_connection_speeds,accessedMay
2. Ahmed, A., Ahmed, E.: A survey on mobile edge computing. In: International Conference on Intelligent Systems and Control (2016)
3. Albers, S.: Energy-efficient algorithms. Commun. ACM **53**(5), 86–96 (2010)
4. Dinh, T.Q., Tang, J., La, Q.D., Quek, T.Q.S.: Offloading in mobile edge computing: task allocation and computational frequency scaling. IEEE Trans. Commun. **65**(8), 3571–3584 (2017). https://doi.org/10.1109/TCOMM.2017.2699660
5. Fan, J., Wang, Z., Xie, Y., Yang, Z.: A theoretical analysis of deep q-learning. In: Learning for Dynamics and Control, pp. 486–489. PMLR (2020)
6. Feng, M., Krunz, M., Zhang, W.: Joint task partitioning and user association for latency minimization in mobile edge computing networks. IEEE Trans. Veh. Technol. **70**(8), 8108–8121 (2021)
7. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: the kitti dataset. Int. J. Robot. Res. (IJRR) (2013)
8. Geng, Y., Yang, Y., Cao, G.: Energy-efficient computation offloading for multicore-based mobile devices. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pp. 46–54. IEEE (2018)

9. Lei, C.: Deep reinforcement learning. In: Lei, C. (ed.) Deep Learning and Practice with MindSpore. CIR, pp. 217–243. Springer, Singapore (2021). https://doi.org/10.1007/978-981-16-2233-5_10
10. Li, Y.: Deep reinforcement learning: an overview. arXiv preprint arXiv:1701.07274 (2017)
11. Lin, X., et al.: A pareto-efficient algorithm for multiple objective optimization in e-commerce recommendation. In: Proceedings of the 13th ACM Conference on Recommender Systems, pp. 20–28 (2019)
12. Liu, L., Chang, Z., Guo, X., Mao, S., Ristaniemi, T.: Multiobjective optimization for computation offloading in fog computing. IEEE Internet Things J. **5**(1), 283–294 (2018). https://doi.org/10.1109/JIOT.2017.2780236
13. Mach, P., Becvar, Z.: MEC: a survey on architecture and computation offloading. IEEE Commun. Surv. Tutor. (2017)
14. Mukherjee, M., Lei, S., Di, W.: Survey of fog computing: fundamental, network applications, and research challenges. IEEE Commun. Surv. Tutor. **20**(3), 1826–1857 (2018)
15. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for VM-based cloudlets in mobile computing. IEEE Pervasive Comput. **8**(4), 14–23 (2009)
16. Sun, Y., Lin, F., Xu, H.: Multi-objective optimization of resource scheduling in fog computing using an improved NSGA-II. Wireless Pers. Commun. **102**(2), 1369–1385 (2018)
17. Sun, Z., Liu, Y., Tao, L.: Attack localization task allocation in wireless sensor networks based on multi-objective binary particle swarm optimization. J. Netw. Comput. Appl. **112**, 29–40 (2018). https://doi.org/10.1016/j.jnca.2018.03.023. https://www.sciencedirect.com/science/article/pii/S1084804518301103
18. Tan, M., et al.: MnasNet: platform-aware neural architecture search for mobile. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2820–2828 (2019)
19. Tolmidis, A.T., Petrou, L.: Multi-objective optimization for dynamic task allocation in a multi-robot system. Eng. Appl. Artif. Intell. **26**(5), 1458–1468 (2013). https://doi.org/10.1016/j.engappai.2013.03.001. https://www.sciencedirect.com/science/article/pii/S0952197613000377
20. Wang, F., Xu, J., Wang, X., Cui, S.: Joint offloading and computing optimization in wireless powered mobile-edge computing systems. IEEE Trans. Wireless Commun. **17**(3), 1784–1797 (2017)
21. Wang, L., Jiao, L., Li, J., Mühlhäuser, M.: Online resource allocation for arbitrary user mobility in distributed edge clouds. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 1281–1290. IEEE (2017)
22. Wang, S., Zhao, Y., Huang, L., Xu, J., Hsu, C.H.: QoS prediction for service recommendations in mobile edge computing. J. Parallel Distrib. Comput. **127**, 134–144 (2019)
23. Wang, S., Li, J., Wu, G., Chen, H., Sun, S.: Joint optimization of task offloading and resource allocation based on differential privacy in vehicular edge computing. IEEE Trans. Comput. Soc. Syst. **9**(1), 109–119 (2021)
24. Xu, Z., et al.: Experience-driven networking: a deep reinforcement learning based approach. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pp. 1871–1879. IEEE (2018)
25. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proceedings of IEEE 36th Annual Foundations of Computer Science, pp. 374–382. IEEE (1995)

26. Yi, S., Qin, Z., Li, Q.: Security and privacy issues of fog computing: a survey. In: Xu, K., Zhu, H. (eds.) WASA 2015. LNCS, vol. 9204, pp. 685–695. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21837-3_67
27. Zhang, F., et al.: A load-aware resource allocation and task scheduling for the emerging cloudlet system. Futur. Gener. Comput. Syst. **87**, 438–456 (2018)
28. Zhang, P., Wang, Y., Kumar, N., Jiang, C., Shi, G.: A security and privacy-preserving approach based on data disturbance for collaborative edge computing in social IoT systems. IEEE Trans. Comput. Soc. Syst. **9**(1), 97–108 (2021)
29. Zhou, J., Zhao, X., Zhang, X., Zhao, D., Li, H.: Task allocation for multi-agent systems based on distributed many-objective evolutionary algorithm and greedy algorithm. IEEE Access **8**, 19306–19318 (2020)
30. Zhou, S., Jadoon, W.: Jointly optimizing offloading decision and bandwidth allocation with energy constraint in mobile edge computing environment. Computing 1–27 (2021)