Cost Efficient Scheduling for Delay-sensitive Tasks in Edge Computing System

Yongchao Zhang*, Xin Chen*, Ying Chen*, Zhuo Li*, Jiwei Huang†
* *School of Computer Science, Beijing Information Science & Technology University*
Beijing, China
Email: zhangyongchao@mail.bistu.edu.cn, {chenxin, chenying, lizhuo}@bistu.edu.cn
† *Beijing University of Posts and Telecommunications*
Beijing, China
Email: huangjw@bupt.edu.cn

***Abstract*—Edge computing, as an emerging computing model, can offload delay-sensitive computing tasks from Internet of Thing (IoT) devices with limited computing resources and energy to the edge cloud. In the edge computing system, several servers are placed on the network edge near the IoT devices to process the offloaded tasks. A key issue in edge computing system is how to reduce the system cost while completing the offloaded tasks. In this paper, we study the task scheduling problem to reduce the cost of edge computing system. We model the task scheduling problem as an optimization problem, where the objective is to minimize the system cost while satisfying the delay requirements of all the tasks. Then, we prove that the proposed optimization problem is NP-hard. To solve this optimization problem effectively, we propose a task scheduling algorithm, called Two-stage Task Scheduling Cost Optimization (TTSCO). We validate the effectiveness of our algorithm by comparing with optimal solutions. The results show that the approximate ratio is less than 1.2 for 95% of the data sets we use. Performance evaluation shows that our algorithm can effectively reduce the cost of edge computing system while satisfying the delay requirements of all the tasks.**

***Index Terms*—edge computing; task scheduling; delay-sensitive tasks; cost efficiency;**

I. Introduction

With the development of IoT technology, the number of delay-sensitive applications (e.g., health monitoring [1], location-based augmented reality games) are increasing rapidly [2]. As the computing resources and energy of IoT devices are limited, many processing-heavy tasks should be offloaded to remote servers to be processed. Cloud computing with powerful computing capacity is considered as a potential way for processing the offloaded tasks. However, due to the long distance between the conventional cloud and IoT devices, sending a large number of tasks to conventional cloud for processing will cause long response time and serious network congestion. To deal with this issue, edge computing is recently proposed as a promising computing model [3], [4]. Edge computing provides an additional layer of computing infrastructure which consists of some servers at the network edge (i.e., base stations). For the computing tasks offloaded from IoT devices, edge computing provides the computing services and returns the results to the devices. In this way, the transmission delay of offloaded task and the traffic load of core network will be greatly reduced. Because of these benefits, edge computing has drawn increasing attention from researchers.

边缘计算的出现

In the edge computing, task scheduling problem has become a hot topic for studying [9]-[18]. To reduce the energy consumption caused by the computing of tasks, IoT devices would offload computing tasks to the edge servers. However, the task offloading consumes extra energy to transfer tasks to the edge servers, and the completion time of offloaded task would also increase. In this case, some works studied how the devices make the task offloading decision [9]-[12]. In addition, when the computing task is scheduled to different edge servers, the cost of transmission and computation are also different. Therefore, some works aimed to reduce the system cost generated by the transmission or computation of tasks [13]-[18]. However, these works rarely considered the cost generated by the edge servers. The problem of reducing the system cost caused by the servers during the off-peak times (e.g., at night) is largely unexplored [5].

问题的出现

In this paper, we study the cost optimization for task scheduling problem in the edge computing system. The goal is to minimize the cost of edge computing system while satisfying the QoS requirements of all the tasks. We develop a task scheduling model for the delay-sensitive input tasks. Then, we formulate the cost optimization problem and prove this optimization problem to be NP-Hard. Next, we propose an approximate algorithm to solve this optimization problem, which is called Two-stage Task Scheduling Cost Optimization (TTSCO), and conduct the analysis for the TTSCO algorithm. Finally, simulation results are provided to verify the accuracy and performance improvement of our algorithm.

The remainder of this paper is as follows. In section II, we discuss the related work in this field. In section III, the system model and the optimization problem are presented. Section VI proposes the TTSCO algorithm, and conducts the analysis. In section VII, we present the simulation results to evaluate the performance of our algorithm. Section VIII concludes this paper and discusses the future work.

II. Related Work

Edge computing has attracted significant attention in recent years. Previous work about the edge computing included fog

2474-2473/18/$31.00 ©2018 IEEE
DOI 10.1109/SCC.2018.00017

computing [6], cloudlet [7], [8], femtoclouds [9], and mobile edge computing [10]. Bonomi et al. [6] introduced the concept and characteristics of fog computing which enabled a style of applications and services. Verbelen et al. [8] presented a cloudlet architecture with three layers, and executed the AR use case based on their architecture. A femtocloud system consisting of several co-located mobile devices was proposed to provide cloud services at the edge [9]. Michael et al. [10] presented an architectural topology of mobile edge computing and classified serval applications deployed at the mobile edge.

There were extensive research studies about the task scheduling problem in edge computing system. Kim et al. [11] studied the cost-delay tradeoff for dual-side in mobile computing system, and proposed control algorithm to minimize the cost in the competition and cooperation scenario. Mao et al. [12] proposed a dynamic computation offloading policy to minimize the execution cost of a green Mobile Edge Computing (MEC) system with energy harvesting devices. Lyu et al. [13] proposed an asymptotically optimal offloading approach to maximize the network utility and guarantee the quality of service of wireless applications. An online joint radio and computational resource management algorithm were proposed to minimize the long-term average power consumption for the multi-user MEC system [14]. These works considered the distribution of workload between the user and edge cloud, but the task scheduling within the edge cloud was not studied.

Jia et al. [15] studied multiple cloudlets placement problem in a large scale Wireless Metropolitan Area Network (WMAN), which was to minimize the average access delay between mobile and cloudlets. Zeng et al. [16] proposed an task scheduling and resource management strategy to minimize the task completion time in fog computing system. Gu et al. [17] proposed a two-phase linear programming-based algorithm to tackle the cost efficiency problem in fog computing medical cyber-physical systems (FC-MCPS). In order to handle the peak workloads from mobile users and efficiently utilize cloud resources, Tong et al. [18] proposed a tree hierarchy of geo-distributed servers in the edge cloud. And they also proposed an efficient heuristic algorithm to schedule the workloads. Urgaonkar et al. [19] modeled the optimize operational costs as a Markov Decision Problem (MDP), and presented an approach to minimize the operational costs in this model. To satisfy the quality of service requirements of the tasks, Song et al. [20] presented an periodical task distribution to maximize the number of tasks which can be processed in the edge cloud. In this works, the task scheduling and resource management problem within the edge cloud were well studied. However, they did not consider the task scheduling problem for reducing system cost generated by the edge servers.

III. SYSTEM MODEL

In this section, we first introduce the edge computing system. Then, based on the developed system model, we formulate cost optimization problem.

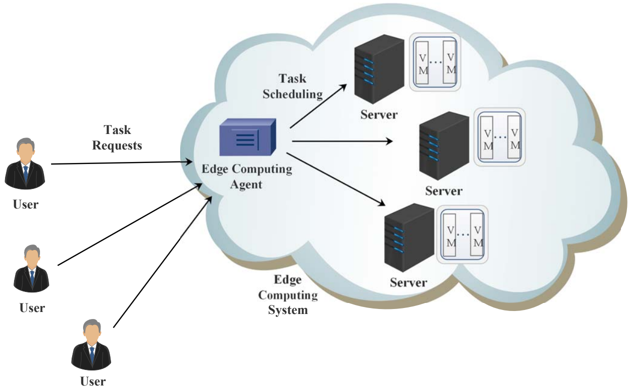

Fig. 1. Edge Computing Architecture.

A. Architecture

An edge computing system is composed of an Edge Computing Agent (ECA) and several heterogeneous edge servers as shown in Fig. 1. The ECA has a global overview of all available resources, and communicates with each deployed server [21]. Each server runs several virtual machines which processes the offloaded tasks on behalf of the users. By offloading the computing tasks to the edge computing system, better quality of experience (i.e., powerful computing capability, low latency) can be achieved.

For each computing task offloaded from the user, ECA will select an available server to process according to its resource requirements. In this paper, we consider that ECA executes task scheduling strategy periodically [20], and the scheduling strategy is executed after the group of current tasks are all completed. Let I be the interval time between two consecutive task scheduling process. As the completion time of each group will be different from each other, we dynamically adjust the value of I, which is given as follows,

$$I = t_{md} \tag{1}$$

Here, t_{md} is the maximum of expect completion time of all the tasks.

B. Problem Formulation

To formulate the task scheduling problem in the edge computing system, we first model the input tasks offloaded from the users and the available edge servers. Then, we derive the resource constraints of task scheduling problem, and propose a unified optimization framework.

1) Task Model and Server Model: In the edge computing system, we consider a group of delay-sensitive tasks $T = \{t_1, t_2, \cdots, t_n\}$ to be processed at the time of executing the tasks scheduling strategy, where n is the total number of input tasks. Each task $t_i \in T$ is indicated by $t_i = \{d_i, w_i, \delta_i, s_i\}$. Let d_i denote the transmission data size of t_i, which includes input data to be computed by the server and output data returned to the users, w_i denote the the computation workload [22] of

TABLE I
NOTATIONS AND DEFINITIONS

Notion	Definition
n	Number of tasks
m	Number of servers
x_{ij}	Denote whether task t_i is scheduled to server e_j
y_j	The state of e_j
t_i	The ith task
e_j	The jth server
s_i	The storage requirement of t_i
δ_i	The dealy requirement of t_i
w_i	The computation workload of t_i
d_i	The size of transmission data of t_i
b_{ij}	The requirement bandwidth if t_i was computed on e_j
S_j	The available storage of server e_j
V_j	The number of VMs that are deployed on server e_j
R_j	The computing rate of each VM under the server e_j
B_j	The bandwidth between e_j and the ECA
C_j	The cost of server e_j

t_i, δ_i denote the deadline requirement of t_i, and s_i denote the storage requirement of t_i.

We consider there are m heterogeneous edge servers $E = \{e_1, e_2, \cdots, e_m\}$ in the edge computing system. Each server $e_j \in E$ is denoted by $e_j = \{B_j, V_j, R_j, S_j\}$. Let B_j denote the available communication bandwidth between server e_j and the ECA at the time of executing task scheduling strategy. A set of VMs are deployed on each server, and each VM can only process one computing task at the same time [20]. Let V_j be the number of VMs that are deployed on server e_j, and S_j denote the available storage resource of server e_j. The computing rate of each VM on the server e_j is the same, which is denoted by R_j. And the bandwidth occupied by each VM can be dynamically adjusted when the task scheduling happens.

Besides, we further denote b_{ij} as the bandwidth requirement when t_i is scheduled to e_j for processing, and C_j as the cost of server e_j when it is in the ON state. For the brevity, the major notations and definitions used in this paper are listed in Table I.

We define binary variable x_{ij} to denote whether task t_i is scheduled to server e_j, i.e.,

$$x_{ij} = \begin{cases} 1, & \text{task } t_i \text{ is scheduled to the server } e_j \\ 0, & \text{otherwise.} \end{cases}$$

Since the edge computing system has sufficient service capacity to process all the task requests, each task t_i will be scheduled to one server for processing. And a task can only be processed by one server. This leads to

$$\sum_{j=1}^{m} x_{ij} = 1 \tag{2}$$

As the resources in each server are limited, these following task scheduling constraints must be satisfied. In each server, there must be sufficient storage space to store the processed tasks, otherwise it will result in the loss of task data. Therefore, the total storage requirement of each task scheduled to server e_j can not exceed the storage resource of server e_j. That is,

$$\sum_{i=1}^{n} x_{ij} s_i \leq S_j \tag{3}$$

In addition, since the number of VMs deployed per server is limited, the total number of tasks scheduled to server e_j can not exceed the number of VMs on server e_j.

$$\sum_{i=1}^{n} x_{ij} \leq V_j \tag{4}$$

2) Performance: In this paper, since the offloaded task has its delay requirement, the constraint is that each task must be completed within its deadline. Basically, the task completion time consists of three basic components: the computation time denoted by l_{ij}^{com}, the input data's transmission time l_{ij}^{in} of t_i from ECA to server e_j, and the output data's transmission time l_{ij}^{out} from server to ECA. Hence, we have:

$$x_{ij}(l_{ij}^{com} + l_{ij}^{in} + l_{ij}^{out}) \leq \delta_i \tag{5}$$

For each task t_i scheduled to server e_j, the three components of time can be calculated as follows [22],

$$l_{ij}^{com} = \frac{w_i}{R_j} \tag{6}$$

$$l_{ij}^{in} = \frac{d_i^{in}}{b_{ij}^{down}} \tag{7}$$

$$l_{ij}^{out} = \frac{d_i^{out}}{b_{ij}^{up}} \tag{8}$$

where d_i^{in} denote the size of task t_i's input data, d_i^{out} denote the size of task t_i's output data, b_{ij}^{up} and b_{ij}^{down} denote the uplink and downlink bandwidth of t_i between the ECA and the server, respectively.

In this paper, we assume that b_{ij}^{up} is equal to b_{ij}^{down}, i.e.,

$$b_{ij}^{*} = b_{ij}^{up} = b_{ij}^{down} \tag{9}$$

Thus, (5) now becomes

$$x_{ij}(\frac{d_i}{b_{ij}^{*}} + \frac{w_i}{R_j}) \leq \delta_i \tag{10}$$

where

$$d_i = d_i^{in} + d_i^{out}$$

If task t_i is scheduled to server e_j, the required bandwidth can be obtained by solving (10). The bandwidth requirement inequality is as follows,

$$b_{ij}^{*} \geq \frac{d_i}{\delta_i - \frac{w_i}{R_j}} \tag{11}$$

To reduce the cost of edge computing system, we consider that all tasks are completed correctly at their respective deadlines δ_i. Then, the required bandwidth of task t_i between the ECA and e_j can be obtained as follows,

$$b_{ij} = \frac{d_i}{\delta_i - \frac{w_i}{R_j}} \tag{12}$$

For each edge server e_j, the sum of required bandwidth of the task scheduled to server e_j must be no greater than the bandwidth of server e_j. That is,

$$\sum_{i=1}^{n} x_{ij} b_{ij} \leq B_j \tag{13}$$

3) Cost: In the edge computing system, since ECA communicates with each edge server and manages it, each server can be switched ON or OFF by the ECA. For the server which is in ON state, the edge computing system will pay a certain cost to maintain the normal running of this server [23]. For example, the resource in ECA will be consumed to manage the running server, and the running server will consume a lot of energy. Let C_j denote the cost which the edge computing system pays when the server e_j is in ON state. In this paper, we define the cost of edge computing system as the sum cost of those servers which are in ON state.

Then we identify the binary variable about the state of e_j. Let y_j represent the state of server e_j. That is,

$$y_j = \begin{cases} 1, & \text{server } e_j \text{ is in ON state} \\ 0, & \text{otherwise.} \end{cases}$$

4) Optimization Problem: In the edge computing system, ECA schedules each offloaded task to the edge server. When the server is assigned tasks, it will be in ON state, otherwise be in OFF state [24]. The optimization objective of our task scheduling strategy is to minimize the cost of the edge computing system.

Summarize all issues discussed above, the cost minimization optimization problem of edge computing system can be formulated as follows,

$$\min_{x_{ij}} \sum_{j=1}^{m} y_j C_j \tag{14}$$

Subject to

$$\sum_{j=1}^{m} x_{ij} = 1 \quad \forall i \in 1, 2, \cdots, n \tag{15}$$

$$\sum_{i=1}^{n} x_{ij} s_i \leq S_j \quad \forall j \in 1, 2, \cdots, m \tag{16}$$

$$\sum_{i=1}^{n} x_{ij} \leq V_j \quad \forall j \in 1, 2, \cdots, m \tag{17}$$

$$\sum_{i=1}^{n} x_{ij} b_{ij} \leq B_j \quad \forall j \in 1, 2, \cdots, m \tag{18}$$

Next, we prove that the cost optimization problem proposed in this paper is NP-hard.

Theorem. *The cost optimization problem formulated by this model is NP-hard.*

Proof. The resources of each edge server e_j can be presented by vector $\vec{p}_{e_j} = (S_j, V_j, B_j)$. When task t_i is computed on server e_j, the resources required by task t_i can be presented by vector $\vec{p}_{t_{ij}} = (s_i, 1, b_{ij})$. We consider a special case of our problem in which all servers are homogeneous. The resources required by task t_i can be rewritten as $\vec{p}_{t_i} = (s_i, 1, b_i)$, and the resources of each server e_j will switch to $\vec{p}_e = (S, V, B)$. Recall the objective of the cost optimization problem is to minimize system cost while ensuring the QoS requirements of all tasks. We can consider each task as a item and each server as a bin. Then, the goal is to use the minimum number of bins to pack all items into bins. Obviously, this special problem is equivalent to the three-dimensional vector packing problem, which is NP-hard [25].

Prove that our cost optimization problem is NP-hard by using reduction to absurdity. Assuming the optimization problem is not NP-hard, then the special case of the problem must not be NP-hard. However, it is contrary to the conclusion we have derived. Hence our cost optimization problem is NP-hard. □

IV. ALGORITHM DESIGN

In this section, we propose an efficient TTSCO algorithm to solve our cost optimization problem. Then, we conduct the analysis for the TTSCO algorithm.

Since the cost optimization problem proposed in this paper is NP-hard, it is impossible to obtain the optimal solution in polynomial time [26]. Based on the heuristic algorithm (BF), we design TTSCO algorithm to solve the optimization problem. The TTSCO algorithm consists of two stages. In the stage 1, we obtain a preliminary task scheduling strategy using the improved BF algorithm. And in the stage 2, we optimize the previous task scheduling scheme and get the finial strategy.

Stage 1.

Since the edge servers are heterogeneous in the cost optimization problem, the TTSCO algorithm should consider how to select edge servers to process the offloaded tasks. Recall the goal of our task scheduling strategy is to minimize the cost of edge computing system, TTSCO algorithm selects the server with the smallest unit cost u_j, which is expressed as (19). Without loss of generality, the unit cost of different edge servers are also unequal.

$$u_j = \frac{C_j}{z_j} \tag{19}$$

Where z_j denote the size of server e_j,

$$z_j = \frac{mS_j}{\sum_{k=1}^{m} S_j} + \frac{mV_k}{\sum_{k=1}^{m} V_k} + \frac{mB_j}{\sum_{k=1}^{m} B_k} \tag{20}$$

We define $\vec{q}_{e_j}$ as the remaining available resources of server e_j after some tasks are scheduled to it. Based on the BF algorithm, for the offloaded tasks which can be processed on the server e_j, ECA selects the *largest* task to be process on this server. In the TTSCO algorithm, we define *largest* to be the task that maximizes the dot product h_i [27]. Formally, the dot product h_i can be expressed as,

$$\begin{aligned} h_i &= \vec{q}_{e_j} \vec{p}_{t_{ij}} \\ &= s_i S'_j + V'_j + b_{ij} B'_j \end{aligned} \tag{21}$$

Where

$$\begin{aligned}\vec{q}_{e_j} &= (S'_j, V'_j, B'_j) \\ &= \vec{p}_{e_j} - \sum p_{t_{*j}} \end{aligned} \quad (22)$$

Within $p_{t_{*j}}$ denote the resource requirements of all tasks scheduled to server e_j.

Stage 2.

After the stage 1, we get the preliminary task scheduling strategy. But it is worth noting that, in some special cases, the system cost of preliminary scheduling strategy can be reduced further.

For a simple example, consider there are two tasks A1, A2 to be computed, and two edge servers B1, B2. The resource requirement vector of tasks A1, A2 are $(10, 10, 10)$ and $(20, 20, 20)$, respectively. And the available resource vector of servers B1 and B2 are $(15, 15, 15)$ and $(50, 50, 50)$, respectively. According to the first scheduling stage, task A1 will be scheduled to server B1, then task A2 will be scheduled to server B2. In this case, the total cost is the sum of two servers' cost. However, if all tasks are scheduled to server B2 for processing, the total cost is only the cost of server B2. By this scheduling strategy, the total cost is obviously reduced.

According to the above example, for the server which is selected in the end, only a small amount resources are used. Therefore, after the first scheduling stage, we need to further optimize the scheduling strategy to reduce unnecessary cost. In the general case, the last selected server will have a large amount of available resources. In the TTSCO algorithm, we propose an optimization strategy to maximize resources utilization of the last selected server. The main idea of optimization strategy is to re-put the tasks in the smallest cost servers into the last selected server after the first stage. In this way, the TTSCO algorithm can increase the utilization of the server with highest cost, and reduce unnecessary cost which are caused by the servers with smaller cost.

Summarize all above discussions, we propose the TTSCO algorithm to solve our optimization problem, as shown in Algorithm 1. The input of the algorithm are the set of tasks to be scheduled to the servers and the set of available edge servers. In line 1, we first initialize the decision binary variable x_{ij} and y_j in the optimization problem.

In stage 1, we sort the edge servers with non-decreasing order of unit cost u_j and non-decreasing order of C_j when the unit costs are equal according to (19) and (20). From line 5 to 16, we select server to process unscheduled tasks based on their resource requirements and the available resources of server. In line 6, we select the server e_j with the smallest unit cost in the set of available servers E. If the available resource of selected server can meet the resource requirements of some tasks in the unscheduled tasks set T, then choose the biggest task to put into server e_j (line 7-14). Otherwise, the server e_j is removed from the set of available servers E (line 15). The biggest task t_i is obtained according to (21) and (22) in line 8. Then task t_i is removed form the set of unscheduled tasks set T, and t_{ij}, y_j are set to be 1 (line 10-12). And the server e_j is added into the set of servers that are used U (line 13).

Algorithm 1 Our TTSCO algorithm for the cost optimization problem

```
Input: Set of tasks to be scheduled $T$, set of available edge servers $E$
Output: Task scheduling variable $\{x_{ij}\}$, the state of servers $\{y_j\}$
 1: Initialize all variable $x_{ij}$ to be 0, set $y_j = 0$ for all servers
 2: Stage 1:
 3: Obtain the vector $\vec{p}_{e_j}$ and compute the unit cost $u_j$ of each server $e_j \in E$
 4: Set of servers that are used $U = \{\}$
 5: while $T \neq \emptyset$ do
 6:     Choose the server $e_j$ with the smallest value of unit cost $u_j$ in $E$
 7:     while there are tasks can be assigned into server $e_j$ do
 8:         Compute the dot product $h$ of all tasks can be assigned into server $e_j$
 9:         Accommodate task $t_i$ with the biggest value of dot product into server $e_j$
10:         $T \leftarrow T \setminus \{t_i\}$
11:         Set $x_{ij} = 1$
12:         Set $y_j = 1$
13:         $U = U \cup \{e_j\}$
14:     end while
15:     $E \leftarrow E \setminus \{e_j\}$
16: end while
17: Stage 2:
18: Obtain the last selected server $e_{g1}$ and server $e_{g2}$ with smallest cost in $U$
19: while task $t_i$ in $e_{g2}$ can be put into the server $e_{g1}$ do
20:     Put task $t_i$ into server $e_{g2}$
21:     Set $x_{ig1} = 1$
22:     if there is no task in $e_{g2}$ then
23:         $U \leftarrow U \setminus \{e_{g2}\}$
24:         Set $y_{g2} = 0$
25:         Obtain the new server $e_{g2}$ with smallest cost in $U$
26:     end if
27: end while
```

In stage 2, we first obtain the last selected server e_{g1} and server e_{g2} with smallest cost in U. If the available resources in server e_{g1} can meet the resource requirements of task t_i in server e_{g2}, we re-put task t_i into server e_{g1} and update the task scheduling variable (line 19-27). When tasks in server e_{g2} are all removed, the server e_{g2} will be removed from U, and update the state of server e_{g2}. Then the new server with the smallest cost is got from U (line 22-25).

We conduct the time complexity analysis for the TTSCO algorithm. Recall the first stage in our TTSCO algorithm, there are two main phases: sorting the servers and scheduling tasks to servers. In the first phase, the process of sorting servers will take at least $O(mlog(m))$ time. In the second phase, there are

n tasks that need to be scheduled. Considering the worst case that the selected server can meet the source requirements of all unscheduled tasks, algorithm will cost n time to compute dot product and choose the biggest task. In this case, the second phase will cost at least n^2 time. Therefore, the worst time complexity of the first stage is $mlog(m) + n^2$. For the second stage, it will task $n - 1$ time in the worst case. So the worst time complexity of our TTSCO algorithm is $mlog(m) + n^2$.

V. Performance Evaluation

In this section, we conduct the performance evaluation of TTSCO algorithm for solving the cost optimization problem. By the simulation results, we show the effectiveness and improvement of our task scheduling strategy, compared with a baseline strategy: random scheduling strategy (RANDOM). The random strategy randomly schedules each task to an available server for processing, where the selected server can satisfy the QoS requirements of task requests.

In the experiments, we consider that the edge computing system is composed of two types of servers. The edge computing system has sufficient service capacity to process all the task requests. For each task request, its delay requirements is as d_i/α, where α is randomly distributed in $[9, 11]$. Without loss of generality, the service capacity parameters of different servers are all different, which include the storage capacity, the number of VMs, the computing rate and communication bandwidth. And the cost of each server is set to be positively related to its service capacity.

A. Comparison Experiment

We first check the performance of results obtained by our TTSCO algorithm. We compare the results of TTSCO algorithm with the optimal solutions. We use LINGO software to obtain the optimal solutions. We plot the results of our algorithm and optimal solutions in Fig. 2. Comparison result shows that the TTSCO algorithm solutions are close to the optimal solution.

Then we denote c_1 as the solution obtained by our TTSCO algorithm, and c_2 as the optimal solution obtained by LINGO software. Let the approximate ratio, c_1/c_2, denote as the indicator of the performance of our algorithm. We plot the cumulatively distribution function (CDF) of the approximation ratio in Fig. 3.

It can be seen form Fig. 3 that the approximate ratio is between 1 and 1.3, and 95% of approximate ratio is less than 1.2. The simulation result can validate the accuracy of our TTSCO algorithm. With the increase of the number of input tasks, we find that a long time (maybe more than one hour) will be spent to get the optimal solutions by LINGO software. However, the approximate solutions of our TTSCO algorithm can be obtained in a few seconds by MATLAB. In addition, since the cost optimization problem is NP-hard, we can not obtain the optimal solutions in the polynomial time. However, an approximate solution can be obtained by our TTSCO algorithm in the polynomial time. Therefore, the TTSCO algorithm proposed in this paper is both accurate and efficient.

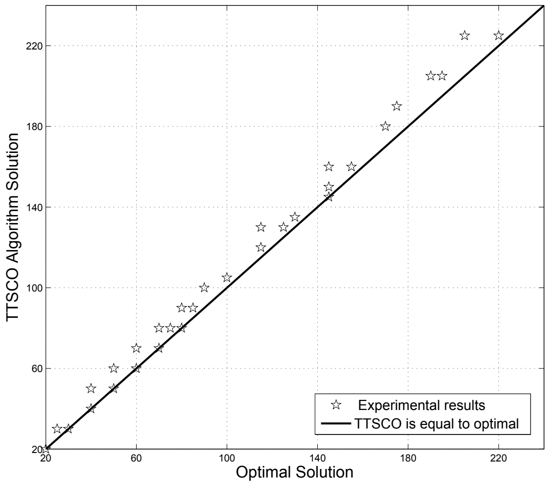

Fig. 2. TTSCO Algorithm and Optimal Solution

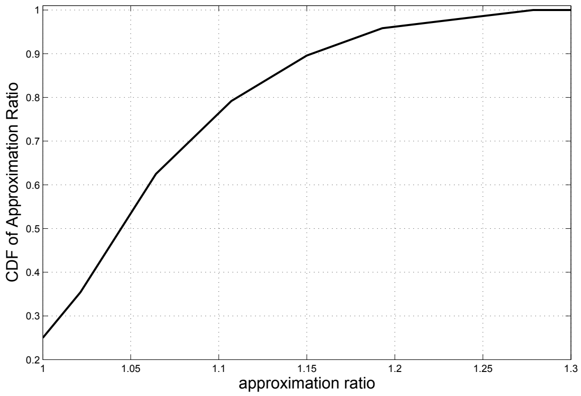

Fig. 3. CDF of approximation ratio

B. Parameters Analysis

In this subsection, we analyse the effect of parameters on our TTSCO algorithm compared with the RANDOM algorithm. These parameters include the number of input tasks, transmission data size and delay requirement.

1) The Effect of Number of Tasks: In this simulation, we evaluate the performance of our TTSCO algorithm with different number of input tasks. The transmission data size of input tasks is set to 50MB. We change the number of input tasks from 30 to 150 with an increment of 30. The performance of TTSCO algorithm is compared with the RANDOM algorithm. We plot the cost of the edge computing system obtained by both algorithm in Fig. 4.

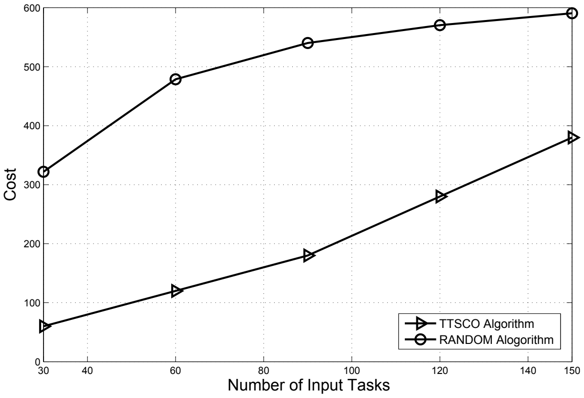

Fig. 4. The Effect of Number of tasks

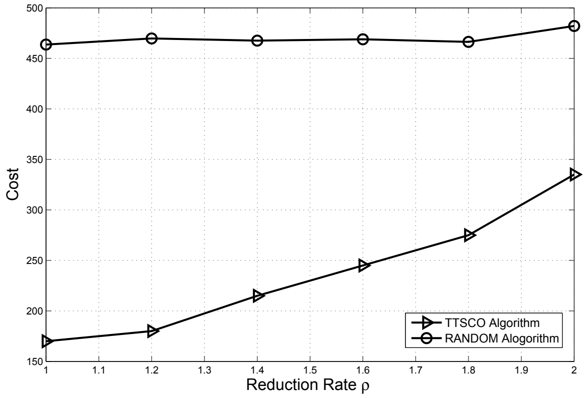

Fig. 6. The Effect of Delay Requirement of Tasks

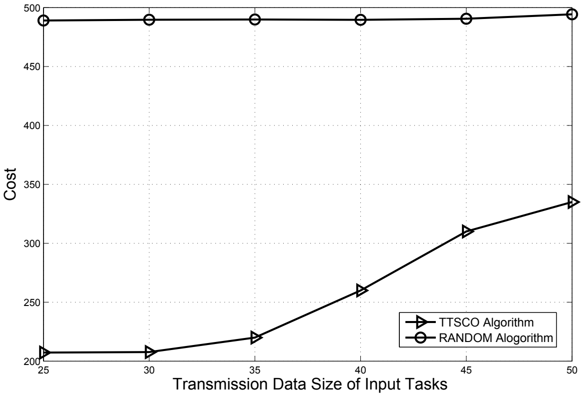

Fig. 5. The Effect of Transmission Data Size

Fig. 4 shows that the cost of edge computing system will increase with the increase of the number of input tasks. This is because more servers are need to process tasks, which leads to increase in the cost. As seen from Fig. 4, our TTSCO algorithm can reduce 50% cost obtained by RANDOM algorithm on average. The reason is that our TTSCO algorithm increases the resource utilization of each used server. In this case, the number of servers in the ON state will decrease.

2) The Effect of Transmission Data Size: In this simulation, we study the impact of different transmission data size of input tasks on the performance of our TTSCO algorithm and RANDOM algorithm. We set the number of input tasks to 150. And the transmission data size of input tasks increases from 25MB to 50MB with an increment of 5MB. We plot the cost of the edge computing system obtained by both algorithms in Fig. 5.

Fig. 5 shows that the cost of edge computing system will generally increase with the increase of transmission data size of tasks under our TTSCO algorithm. However, when the transmission data size of tasks is small (i.e. below 30MB), the cost will not increase with the increase of transmission data size. The reason is that the number of VMs is the main limitation of task scheduling. However, the resource utilization of bandwidth and storage on each server is relatively low. When the transmission data size of tasks increases, resource utilization of each server will increase, but the total cost will not change (i.e. between 25MB and 30MB). When the transmission data size continues to increase, the resources of the used servers are not enough to process these tasks, and more servers need to be used. In this case, the system cost will increase with the increase of transmission data size.

It can also be seen from Fig. 5 that the cost is roughly the same by the RANDOM algorithm. The reason is that the RANDOM algorithm randomly selects the server to process tasks, which causes lots of running servers are with low resource utilization. With the increase of transmission data size, resource utilization of each server increases, but the total cost will remain unchanged. It is showed in Fig. 5 that our TTSSCO algorithm can reduce 50% cost obtained by RANDOM algorithm on average.

3) The Effect of Delay Requirement of Tasks: In this simulation, we evaluate the performance of our TTSCO algorithm with the different delay requirement of input tasks. We define a task set including 7 tasks where the transmission data size of tasks is varied from 20MB to 50MB with an increment of 5MB. We use 15 task sets as the input tasks. The delay requirement of each task is set to δ/ρ, where δ is the original delay and ρ is the reduction rate. We change ρ from 1 to 2 with an increment of 0.2, and plot the cost of the edge computing system obtained by both algorithms in Fig. 6.

Similar result can be drawn from Fig. 6. With the increase of reduction rate ρ, the cost obtained by TTSCO algorithm will increase, and the cost by RANDOM strategy is not changed basically. This is because the smaller delay means the larger transmission bandwidth is required, that causes more servers need to be used to process input tasks. Fig. 6 shows that our TTSCO algorithm can reduce 45% cost that is obtained by RANDOM algorithm on the average.

VI. Conclusion

In this paper, we explore the cost optimization for task scheduling problem in the edge computing system. We propose the optimization problem to minimize the cost of the edge computing system while satisfying all input tasks' QoS requirements. An effective algorithm, called TTSCO algorithm, is proposed to solve the cost optimization problem. By comparing with optimal solutions obtained by LINGO software, we verify the accuracy of our TTSCO algorithm. Performance evaluation results show the improvement of our algorithm in reducing the cost of edge computing system compared with the RANDOM algorithm. In the future research, we will consider dynamic resource management and tasking scheduling among multiple edge clouds.

Acknowledgment

This work is partly supported by the National Natural Science Foundation of China (No.61370065, 61502040), Supplementary and Supportive Project for Teachers at Beijing Information Science and Technology University (No.5111823401), National Key Technology Research and Development Program of the Ministry of Science and Technology of China (No.2015BAK12B03-03) and Beijing Municipal Program for Excellent Teacher Promotion (No.PXM2017_014224_000028).

References

[1] M Hassanalieragh et al., ”Health Monitoring and Management Using Internet-of-Things (IoT) Sensing with Cloud-Based Processing: Opportunities and Challenges.” IEEE International Conference on Services Computing, 2015:285-292.

[2] S Tata, R Jain, H Ludwig, S Gopisetty, ”Living in the Cloud or on the Edge: Opportunities and Challenges of IOT Application Architecture.” IEEE International Conference on Services Computing, 2017:220-224.

[3] W. Shi et al., Edge Computing: Vision and Challenges, IEEE Internet of Things J., vol. 3, no. 5, 2016, pp. 637-646.

[4] S Li, J Huang, ”GSPN-Based Reliability-Aware Performance Evaluation of IoT Services.” IEEE International Conference on Services Computing, 2017:483-486.

[5] Y.-J. Ku et al., 5G radio access network design with the fog paradigm: Confluence of communications and computing, IEEE Commun. Mag., vol. 55, no. 4, pp. 46-52, Apr. 2017.

[6] F. Bonomi, ”Fog Computing and Its Role in the Internet of Things”, Proc. First Edition of the MCC Workshop on Mobile Cloud Computing, Aug. 2018.

[7] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, ”The case for vm-based cloudlets in mobile computing”, IEEE Pervasive Computing, vol. 8, no. 4, pp. 14-23, Oct 2009.

[8] T. Verbelen and P. Simoens and F. De Turck and B. Dhoedt, ”Cloudlets: Bringing the Cloud to the Mobile User,” in Proc. of ACM Workshop on Mobile Cloud Computing and Services, 2012, pp. 29-36.

[9] K. Habak, M. Ammar, K. A. Harras, E. Zegura, ”Femto clouds: Leveraging mobile devices to provide cloud service at the edge”, Cloud Computing (CLOUD) 2015 IEEE 8th International Conference, pp. 9-16, 2015.

[10] M. T. Beck, M. Werner, S. Feld, and T. Schimper, ”Mobile edge computing: A taxonomy, ” in Proc. of the Sixth International Conference on Advances in Future Internet, AFIN, 2014.

[11] Kim, Yeongjin, J. Kwak, and C. Song. Dual-side Optimization for Cost-Delay Tradeoff in Mobile Edge Computing. IEEE Transactions on Vehicular Technology pp. 99 :1-1, 2017.

[12] Y. Mao, J. Zhang, and K. B. Letaief, Dynamic computation offloading for mobile-edge computing with energy harvesting devices, IEEE Journal on Selected Areas in Communications, vol. 34, no. 12, pp. 3590-3605, 2016.

[13] X. Lyu, W. Ni, H. Tian, R. P. Liu, X. Wang, G. B. Giannakis, and A. Paulraj, Optimal schedule of mobile edge computing for Internet of Things using partial information, IEEE J. Sel. Areas Commun., accepted, 2017.

[14] Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief, Stochastic joint radio and computational resource management for multi-user mobileedge computing systems, 2017, http://arxiv.org/abs/1702.00892.

[15] M. Jia, J. Cao, and W. Liang. Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. IEEE Transactions on Cloud Computing, in press, 2015.

[16] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, Joint optimization of task scheduling and image placement in fog computing supported softwaredefined embedded system, IEEE Transactions on Computers, vol. 65, no. 12, pp. 3702-3712, 2016.

[17] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, Cost efficient resource management in fog computing supported medical cyberphysical system, IEEE Transactions on Emerging Topics in Computing, vol. 5, no. 1, pp. 108-119, 2017.

[18] L. Tong, Y. Li, and W. Gao, A hierarchical edge cloud architecture for mobile computing, in IEEE INFOCOM16, San Francisco, CA, USA, July 2016, pp. 1-9.

[19] R. Urgaonkar, S. Wang, T. Hea, M. Zafer, K. Chan, and K. K. Leung, Dynamic service migration and workload scheduling in edgeclouds, Performance Evaluation, vol. 91, pp. 205-228, 2015.

[20] Song Y, Yau S S, Yu R, et al. ”An Approach to QoS-based Task Distribution in Edge Computing Networks for IoT Applications.” IEEE International Conference on Edge Computing IEEE, 2017:32-39.

[21] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. 2012. Cloudlets: bringing the cloud to the mobile user. In Proceedings of the third ACM workshop on Mobile cloud computing and services (MCS ’12). ACM, New York, NY, USA, 29-36.

[22] K. Kumar, J. Liu, Y. Lu, and B. K. Bhargava, A survey of computation offloading for mobile systems, Mobile Networks and Applications, vol. 18, no. 1, pp. 129-140, 2013.

[23] Hadji, Makhlouf, and D. Zeghlache. ”Minimum Cost Maximum Flow Algorithm for Dynamic Resource Allocation in Clouds.” IEEE, International Conference on Cloud Computing IEEE, 2012:876-882.

[24] F. Cao, M.M. Zhu, C.Q. Wu, ”Energy-efficient resource management for scientific workflows in clouds” in Services (SERVICES) 2014 IEEE World Congress on, IEEE, 2014.

[25] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, Heuristics for vector bin packing. Microsoft Research, Tech. Rep., 2010.

[26] Kang, J, and Park, S, Algorithms for the variable sized bin packing problem, European Journal Operational Research, vol. 147, no. 2, pp. 365-372, 2003.

[27] M. Gabay, S. Zaourar, ”Variable Size Vector Bin Packing Heuristics - Application to the Machine Reassignment Problem”, INRIA Tech. Rep., 2013.