TD-EUA: Task-Decomposable Edge User Allocation with QoE Optimization

Guobing Zou[1,2], Ya Liu[1], Zhen Qin[1(✉)], Jin Chen[1], Zhiwei Xu[1], Yanglan Gan[3], Bofeng Zhang[1], and Qiang He[4(✉)]

[1] School of Computer Engineering and Science, Shanghai University, Shanghai, China
{gbzou,ambersoul,zhenqin,cj1125,zhiweixu,bfzhang}@shu.edu.cn
[2] Shanghai Key Laboratory of Computer Software Testing and Evaluating, Shanghai, China
[3] School of Computer Science and Technology, Donghua University, Shanghai, China
ylgan@dhu.edu.cn
[4] Department of Computer Science and Software Engineering, Swinburne University of Technology, Melbourne, Australia
qhe@swin.edu.au

Abstract. The edge user allocation (EUA) problem has attracted a lot of attention recently. EUA aims at allocating edge users to nearby edge servers strategically to ensure low-latency network connection. Existing approaches assume that a users' request can only be served by an individual edge server or cannot be served at all. They neglect the fact that a user's request may be decomposable and partitioned into multiple tasks to be performed by different edge servers. To tackle this new task-decomposable edge user allocation (TD-EUA) problem, we model it as an optimization problem. Two novel approaches named TD-EUA-O and TD-EUA-H are proposed, one for finding the optimal solution based on Integer Linear Programming that maximizes users' overall Quality of Experience (QoE), and the other for efficiently finding a sub-optimal solution in large-scale EUA scenarios. Extensive experiments based on a widely-used real-world dataset are conducted to evaluate the effectiveness and efficiency of our approaches. The results demonstrate that our approaches significantly outperform the baseline and the state-of-the-art approach.

Keywords: Edge computing · Edge user allocation · Task decomposition · Quality of Experience · QoE optimization

1 Introduction

The rapidly increasing popularity of mobile and Internet-of-Things (IoT) devices, including mobile phones, wearables, sensors, etc., has promoted the growth of versatile computational-intensive applications, such as face recognition, machine vision, intelligent furniture [6]. Due to the limited computing capabilities and battery power of mobile and IoT devices, their computing tasks are often

E. Kafeza et al. (Eds.): ICSOC 2020, LNCS 12571, pp. 215–231, 2020.
https://doi.org/10.1007/978-3-030-65310-1_17

offloaded to app vendors' servers in the cloud. Nevertheless, with the exponential growth in the number of mobile and IoT devices, it is becoming difficult for cloud computing to handle the huge workload and network congestion. This makes it difficult to provide a low-latency and reliable connection to end-users, especially those that desire real-time responses from applications. For example, delays caused by the traditional centralized computing paradigm may cause operation failures on autopilot and endanger passengers' lives.

To tackle this issue, edge computing, has been proposed as a new distributed computing paradigm [1,15]. In the edge environment, each base station is equipped with a certain amount of computing resources, allowing computing power to be provided to mobile users at the Internet access level. Compared to cloud computing, edge computing places storage and computing resources (such as CPU, memory, bandwidth, etc.,) closer to end-users. An edge server usually cover a specific geographical area [10]. Typically, edge servers are geographically distributed to offer diverse services for different areas. To avoid the existence of an area that is not covered by any edge server, there are overlapping areas between adjacent edges. A user located in the overlapping area can connect to one of the edge servers covering them (*proximity constraint*) that has sufficient computing resources (*resource constraint*) such as CPU, storage, bandwidth, or memory [8,10,11].

While offering new opportunities, edge computing also raises many new challenges, such as the problem of edge user allocation (EUA). As an intermediate supply station, an edge server has limited computing resources. Hence, an app vendor's users in an area must be allocated to edge servers properly to utilize the computing resources hired by the app vendor on the edge servers. Existing research treat each user as a resource request, and each user can only be assigned to one edge server to achieve certain specific optimization objectives, e.g., to minimize the number of edge servers needed [8,10], to maximize the overall user satisfaction measured by their Quality of Experience (QoE) [11] and to increase the ratio of user allocation [14]. However, they ignore the real-world application scenarios where user needs may be decomposable and a user's needs may be satisfied collectively by multiple collaborative edge servers. In a real-world application, a user's request may be composed by multiple tasks, which may need to be performed different edge severs with different resources [20]. Consider a typical game streaming service for example. Players talk to their teammates a lot while playing a game, generating different types of tasks to be performed by edge servers.

The need to handle users' decomposable requests significantly complicates the EUA problem. The fundamental limitation of current EUA approaches assume that a user's needs for computing resources are either fulfilled by an individual edge server nearby, or cannot be fulfilled at all. In this study, we focus on more realistic EUA scenarios where a user's needs may be satisfied by several collaborative edge servers nearby by decomposing its request to a set of tasks to be performed by individual edge servers.

We refer to this problem as a task-decomposable edge user allocation (TD-EUA) problem. To tackle this problem, we model it as an optimization problem and propose two novel approaches, one for finding the optimal solution that maximizes users' overall QoE, and the other for efficiently finding a sub-optimal solution to large-scale TD-EUA problems. To the best of our knowledge, it is the first attempt to tackle the EUA problem where users' requests are decomposable. Our main contributions are as follows:

- We formally define and model the TD-EUA problem, and prove its NP-hardness.
- We propose an optimal approach based on integer linear programming (ILP) for solving the TD-EUA problem that aims to maximize users' overall QoE.
- We propose a heuristic approach for finding a sub-optimal solution to large-scale TD-EUA problems.
- Extensive experiments based on a widely-used real-world dataset are carried out to demonstrate the effectiveness and efficiency of our approaches against a baseline approach and a state-of-the-art approach.

The remainder of the paper is organized as follows. Section 2 provides a motivating example for our research. Section 3 defines and formulates the TD-EUA problem. Section 4 models TD-EUA problem as an optimization problem and presents our approaches in detail. Section 5 shows the experimental evaluation. Section 6 reviews the related work. Finally, we conclude the paper and point out future work in Sect. 7.

2 Motivating Example

A typical example of a task-decomposable EUA application scenario is shown in Fig. 1. In the edge computing environment, there are nine users, $u_1, \cdots, u_9$, four edge server $s_1, \cdots, s_4$, and ten tasks $t_1, t_2, \cdots, t_{10}$, where each task can be performed by a corresponding service deployed on an edge server. Each edge server covers a specific geographical area and has a specific amount of different types of resources available to serve users within its coverage. Edge servers' resource capacities and tasks' resource demand are denoted as a vector $\langle CPU, RAM, storage, bandwidth \rangle$. Each user has a list of tasks and each task may require different amounts of computing resources.

For example, user u_2 has a task list $\{t_2, t_4, t_7, t_8\}$. If the resources available on edge servers s_1, s_2 or s_3 are not limited, user u_2 can be served by any of the three edge servers. Otherwise, the need of u_2 can be partitioned. For example, its tasks t_2, t_4, t_7, t_8 can be served by multiple edge servers. Assume that user u_1 has all the resources it needs from edge server s_2, users u_3 and u_6 are assigned to server s_3, and the workload generated by each task is $\langle 1, 1, 1, 1 \rangle$. As a result, the remaining resources on edge server s_2 or s_3 cannot fulfil the demand of user u_2. Existing EUA approaches cannot handle such case and will allocate user u_2 to the cloud for task processing. However, if the user's requirements can be partitioned, this issue can be addressed. Note that user u_2 is in the overlapping

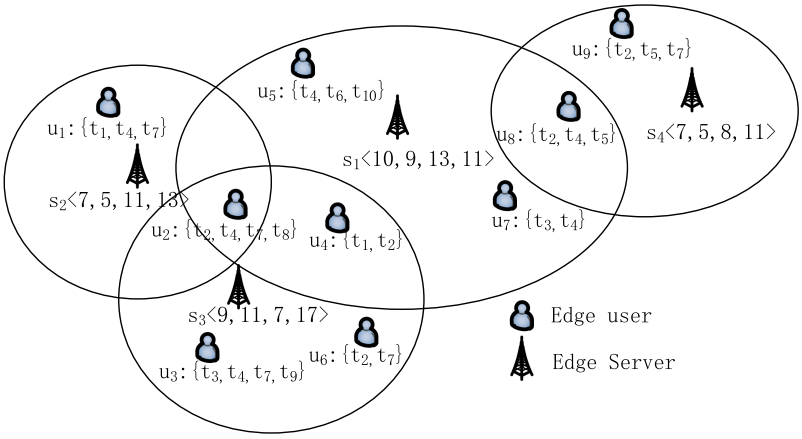


Fig. 1. An example task-decomposable EUA problem.

area of edge servers s_1, s_2, s_3. Tasks t_2 and t_4 can be offloaded to edge server s_2 and task t_7 can be performed by edge server s_3. Then, task t_8 can be performed by edge server s_1. This way, the total workload of the tasks allocated to edge server s_2 is $\langle 5, 5, 5, 5 \rangle$, which not exceed the its remaining capacity ($\langle 7, 5, 11, 13 \rangle$). In the meantime, server s_3 has abundant resources for the tasks assigned to it. What's more, users u_4, u_5, u_7 can be allocated to server s_1 and user u_9 can be allocated to server s_4. Then, user u_8 can allocate its task t_2 to server s_1, tasks t_4 and t_5 to server s_4. While fulfiling the proximity and capacity constraint, this solution allocates all the users' tasks to edge servers, none to the cloud.

There may be other solutions that can also allocate all the users' tasks to edge servers. Finding the optimal one that maximizes users' overall QoE is not trivial, especially in large-scale scenarios. Thus, there is a need for an effective and efficient approach for finding TD-EUA solutions.

3 Problem Formulation

This section defines the TD-EUA problem. The notations and descriptions used in this paper are summarized in Table 1. With the consideration of task decomposition in EUA problem, we give a set of definitions.

Given a finite set of m edge servers $S = \{s_1, s_2, \ldots, s_m\}$, and n edge users $U = \{u_1, u_2, \ldots, u_n\}$ in a particular area, each user has a task list for a request, defined as follows.

Definition 1. *(User Task Decomposition) Given an edge user u, u's request r is composed of a set of tasks, $T(u_i) = \{t_1, t_2, \ldots\}$, where each task t_k can be performed by an edge server.*

From the app vendor's perspective, a TD-EUA solution should allocate as many user requests as possible to edge servers, so that the users' overall QoE is

maximized. A user u_i can only offload one or multiple tasks to an edge server s_j under the condition that it is located within s_j's coverage area $cov(s_j)$.

Definition 2. *(Distance-Aware User Coverage) Given an edge user u_i and a set of edge servers $S = \{s_1, s_2, ..., s_n\}$, only the edge servers that cover user u_i may serve u_i, denoted as $S(u_i)$. The edge servers in $S(u_i)$ fulfil the proximity constraint with respect to user u_i:*

$$d_{ij} \leq cov(s_j), \forall i \in 1, 2, \ldots, n; \forall j \in 1, 2, \ldots, m \tag{1}$$

Table 1. Notations

Notation	Description
$D = \{CPU, RAM, storage, bandwidth\}$	A set of computing resource types
$S = \{s_1, s_2, \ldots, s_m\}$	A set of edge servers
$U = \{u_1, u_2, \ldots, u_n\}$	A set of edge users
$T = \{t_1, t_2, \ldots, t_q\}$	A set of tasks decomposed from users' service requests
$c_j = \langle c_j^1, c_j^2, \ldots, c_j^d \rangle$	Computing capacity of edge server s_j
$w_k = \langle w_k^1, w_k^2, \ldots, w_k^d \rangle$	Computing resources demanded for the task t_k
$W_i = \langle W_i^1, W_i^2, \ldots, W_i^d \rangle$	Computing resource that the user u_i gets from the edge server
$T(u_i)$	A set of tasks which user u_i needs in a service request, $T(u_i) \subseteq T$
$T(s_j)$	A set of tasks allocated to server s_j
$U(s_j)$	A set of users that edge server s_j covers, $U(s_j) \subseteq U$
$S(u_i)$	A set of user u_i's candidate servers - edge servers that cover user u_i, $S(u_i) \subseteq S$
$cov(s_j)$	Coverage radius of edge server s_j
d_{ij}	Geographical distance between user u_i and server s_j

Take Fig. 1 as an example. User u_4 can be served by servers s_1 or s_3. Server s_3 can serve users u_2, u_3, u_4, and u_6 as long as it has adequate resources.

The total workload generated by all the tasks allocated to an edge server must not exceed its current capacity. Otherwise, the server will be overloaded.

Definition 3. *(Server Capacity Constraint) Given an edge server s_j and its covered users $U_c = \{u_c^1, u_c^2, ...\}$, where each user in U_c has a set of tasks. We denote $T(s_j) = \left\{t_{s_j}^1, t_{s_j}^2, ...\right\}$ as the tasks allocated to server s_j, the total resource demand of which must not exceed its current computing capacity:*

$$\sum_{t_{s_j}^k \in T(s_j)} w_k \leq c_j, \forall s_j \in S \tag{2}$$

Take Fig. 1 for an instance, as the workload generated by each task is $\langle 1, 1, 1, 1\rangle$, the aggregate workload incurred by users u_3 and u_6 is $\langle 6, 6, 6, 6\rangle$. It does not exceed the current capacity of server s_3 $\langle 9, 11, 7, 17\rangle$. Therefore, it is a valid allocation. However, if we allocate users u_1 and u_2's tasks t_2, t_4, t_7 to server s_2, it will be overloaded since the aggregate task workload is $\langle 6, 6, 6, 6\rangle$, exceeding server s_2's current computing capacity $\langle 7, 5, 11, 13\rangle$.

Through allocating users' tasks to edge servers, an QoE value can be calculated for each user. In this study, we measure a user's QoE in the same way as [11], which depends on the Quality of the Service (QoS) delivered to the user. As stated in [7,9], QoS is non-linearly correlated with QoE. Generally, it starts to increase slowly at first, then speeds up, and finally converges. Many studies model the correlation between QoE and QoS with the sigmoid function [11]. In [11], the authors use a logistic function, a generalized version of the sigmoid function, to model the QoE-QoS correlation, which is represented as follows:

$$E_i^0 = \frac{L}{1 + e^{-\alpha(x_i - \beta)}} \tag{3}$$

where L is the maximum value of QoE, β is a domain-specific parameter that controls the QoE growth should be, or the mid-point of the QoE function, α, another domain-specific parameter, controls the growth rate of the QoE level, i.e., how steep the change from the minimum to maximum QoE level is, E_i^0 represents the QoE level given user u_i's QoS level W_i, and $x_i = \frac{\sum_{l \in D} W_i^l}{|D|}$. There is $E_i^0 = 0$ if user u_i is not allocated to any edge servers.

Now, we measure the QoE of a user in a TD-EUA scenario, where the its tasks may be allocated to multiple edge servers:

$$E_i = \frac{\sum_{l \in D} W_i^l}{\sum_{l \in D} \sum_{t_k \in T(u_i)} w_k^l} E_i^0 \tag{4}$$

Next, we formally define the TD-EUA problem:

Definition 4. *(TD-EUA) The TD-EUA problem can be represented by a four tuple $TD - EUA =< U, S, T, W >$,where*

(1) $U = \{u_1, u_2, ..., u_n\}$ *is a set of edge users and each user has a request;*
(2) $S = \{s_1, s_2, ..., s_m\}$ *is a set of edge servers, each server has a coverage radius;*
(3) $T = \{t_1, t_2, ..., t_q\}$ *is a set of tasks decomposed from a user's request;*
(4) $W = \{w_1, w_2, ..., w_q\}$ *is a set of resource demands from a task in* T*.*

The solution to a TD-EUA problem is a user-task-server assignment, where the each user's tasks are fully or partially allocated to their nearby edge servers. Based on the assignment, a QoE value can be calculated for each user based on its QoS level. From the app vendor's perspective, its objective is to maximize the users' overall QoE.

4 Approaches

To solve a TD-EUA problem, we first model it as an integer linear programming (ILP) problem to find its optimal solution. To solve large-scale TD-EUA problems efficiently, we propose a heuristic approach named TD-EUA-H that finds a sub-optimal TD-EUA solution.

4.1 Optimal Approach

The optimization objective of TD-EUA is to maximize the users' overall QoE, while satisfying the capacity constraint and proximity constraint. In this section, we present TD-EUA-O, our approach for finding the optimal solution to a TD-EUA problem. It models the TD-EUA problem as an ILP problem as follows:

$$\text{objective function: } \max \sum_{i=1}^{n} E_i \tag{5}$$

s.t.:

$$x_{i,j,k} = 0 \qquad \forall i, j \in \{i, j | d_{ij} > cov(s_j)\}, \forall k \in \{1, 2, \ldots, |T(u_i)\} \tag{6}$$

$$\sum_{i=1}^{n} \sum_{k=1}^{q} w_k x_{i,j,k} \leq c_j \qquad \forall j \in \{1, \ldots, m\} \tag{7}$$

$$\sum_{j=1}^{m} x_{i,j,k} \leq 1 \qquad \forall i \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, q\} \tag{8}$$

where $x_{i,j,k}$ is a binary variable indicating that,

$$x_{i,j,k} = \begin{cases} 1, & \text{if user } u_i\text{'s task } t_k \text{ is allocated to server } s_j \\ 0, & \text{otherwise.} \end{cases}$$

The objective (5) maximizes the users' overall QoE. In (5), QoE level E_i depends on the ratio of the resources W_i the user u_i obtains over the total resources requested by u_i. Constraint (6) enforces the proximity constraint.

A user may be located within the overlapping coverage area of multiple edge servers. Constraint (7) makes sure that the aggregate resource demands of all tasks allocated to an edge server must not exceed that server's current computing capacity. Constraint (8) ensures that each task can be allocated to at most one edge server.

The above ILP problem can be solved by an ILP problem solver, e.g., CPLEX or Gurobi. The outcome is the optimal solution to the TD-EUA problem.

4.2 Problem Hardness

Based on the optimization model, we now prove that the TD-EUA problem is $\mathcal{NP}$-hard.

Theorem 1. *Knapsack* $\leq_p$ *TD-EUA. Therefore, TD-EUA problem is NP-hard.*

Proof. We prove that the TD-EUA problem is $\mathcal{NP}$-hard by reducing the $\mathcal{NP}$-hard Knapsack problem to a specialization of the TD-EUA problem.

Definition 5. *(Knapsack Problem) Given n items and their corresponding weights and values, the aim of a Knapsack problem is to select a group of items so that the total price of the items is the highest within the total weight limit. It can be formally defined as follows:*

$$\textit{objective function:} \max \sum_{i=1}^{n} v_i x_i \tag{9}$$

s.t.:

$$\sum_{i=1}^{n} w_i x_i \leq W \tag{10}$$

$$x_i \in \{0, 1\} \tag{11}$$

where W is the total weight limit and x_i indicates whether the i-th item is selected.

Based on the definition of the Knapsack problem, we now prove that it can be reduced to a special instance of the TD-EUA problem. For ease of exposition, we make the following assumptions: 1) For each task t_k, its requirements for different types of computing resources are the same, i.e., $w_k^1 = w_k^2 = \ldots = w_k^d$; 2) For any edge server s_j, its computing capacities in different dimensions are equal, i.e., $c_j^1 = c_j^2 = \ldots = c_j^d$; 3) The coverage of each edge server is infinite, i.e., a task can be allocated any of the edge server in the area.

Based on the above assumptions, we can obtain a simplified special case of the TD-EUA problem. For the simplified special case, constraints (6)(8) can be combined and projected to (11), because any task can be allocated to any edge server. Moreover, constraint (7) can be projected to objective function (10), since the computing capacities of all the edge servers can be aggregated as an overall resource limit. Clearly, there is a solution to the TD-EUA problem if and only if there is a solution to the corresponding Knapsack problem. Thus, TD-EUA problem is $\mathcal{NP}$-hard.

4.3 Heuristic Approach

Due to the NP-hardness of TD-EUA problem, finding its optimal solution is intractable in large-scale scenarios. This is demonstrated in our experimental results presented in Sect. 5. Thus, we propose a heuristic approach named TD-EUA-H for finding a sub-optimal solution to a TD-EUA problem efficiently. Algorithm 1 presents its pseudo code.

TD-EUA-H goes through three main steps: 1) it employs the skyline algorithm to partition the tasks decomposed from users' requests into two categories, including a group of tasks T_1 requiring more computing resources than any task in the other group T_2; 2) it sorts the tasks within each group according to their required computing resources from high to low; 3) when orderly assigning each task in T_1 to an edge server, for each candidate edge server, it calculates the ratio of the remaining computing resources on that edge server over the number of unallocated tasks covered. Then, it finds the edge server s_j with the highest ratio (Line 15), then allocates the task to that edge server (Lines 13–17). In the same way, it orderly allocates each task t_k in T_2 to an edge server.

The time complexity of TD-EUA-H consists of: 1) using the skyline algorithm to partition q tasks takes $\mathcal{O}(q^2)$ time; 2) labeling and sorting the tasks for each user which depends on the total number of tasks takes $\mathcal{O}(n * q)$ time, where n and q are the number of edge users and tasks, respectively; 3)calculating and

Algorithm 1. TD-EUA-H

Input: edge servers S; edge users U; tasks T.
Output: task-server allocation $f : T \rightarrow S$.

1: $T \xrightarrow{skyline} Good(T), Bad(T)$;
2: **for** each $u_i \in U$ **do**
3: **for** each $t_k \in \mathrm{T}(u_i)$ **do**
4: **if** $t_k \in \mathrm{Good}(T)$ **then**
5: $T_1 \leftarrow (t_k, u_i)$
6: **else**
7: $T_2 \leftarrow (t_k, u_i)$
8: **end if**
9: **end for**
10: **end for**
11: $sort(T_1)_{key} = w_k$, $sort(T_2)_{key} = w_k$
12: **for** each (t_k, u_i) in T_1 **do**
13: $S(u_i) \leftarrow \{s_j \in S | u_i \in cov(s_j)\}$;
14: **if** $S(u_i) \neq \phi$ **then**
15: $j = \mathrm{argmax}\ c_j/\mathrm{unallocated}\ \left(|\sum_{u_i \in U(s_j)} \sum_{t_k \in T(u_i)} t_k|\right)$
16: **end if**
17: $f \leftarrow f \cup \{t_k, s_j\}$
18: **end for**
19: Perform task-server allocation for T_2

ranking m candidate edge servers for each task takes $\mathcal{O}(m \log m)$, and $\mathcal{O}(n * q * m \log m)$ time for all the tasks. Thus, the overall time complexity of TD-EUA-H is $\mathcal{O}(q^2) + \mathcal{O}(n * q) + \mathcal{O}(n * q * m \log m)$. Its complexity indicates that it is an efficient heuristic algorithm with polynomial time for task-decomposable edge user allocation. Thus, it can handle large-scale TD-EUA scenarios.

5 Experiments

5.1 Experimental Setup and Dataset

We conduct a series of experiments to evaluate the effectiveness and efficiency of our approaches. All the experiments are conducted on a machine equipped with an Intel(R) Xeon(R) Gold 6130 CPU@2 and 192 GB RAM. The ILP model in Sect. 4.1 is solved with Gurobi.

The experiments are conducted on the public and widely-used EUA dataset[1]. It contains the locations of the 125 edge servers (base stations) in the Melbourne central business district area in Australia. Following the Gaussian distribution $N(u, \sigma)$, users are distributed in different ways in this area to simulate six different real-world TD-EUA scenarios with different user distributions, as illustrated in Fig. 2, where each black point represents an edge server and each orange point represents a user. Accordingly, six datasets are synthesized with data extracted from the EUA dataset, each corresponding to a specific type of the six user distribution in Fig. 2.

5.2 Competing Methods and Evaluation Metrics

To evaluate the performance of TD-EUA-O and TD-EUA-H, we compare them with two other approaches, including a random baseline and a state-of-the-art approach for solving the EUA problem.

- Random: each task is allocated to a random edge server that has sufficient computing resources to accommodate the task, as long as the user of the task is located within the edge server's coverage area.
- VSVBP [10,11]: it models the EUA problem as a variable sized vector bin packing (VSVBP) problem and aims at maximizing the number of allocated users, while minimizing the number of edge servers needs to be used. This approach treats each user request as a whole, i.e., one user can either be allocated to only one edge server, or cannot be allocated to any edge server at all.

Three widely-used performance metrics are employed in the experiments.

- QoE: it is measured by users' overall QoE, the higher the better.
- Allocation Rate: it is measured by the percentage of users allocated to edge servers of all, the higher the better.
- CPU Time: it is measured by the computation time consumed to find a solution, the lower the better.

[1] https://sites.google.com/site/heqiang/eua-repository. https://github.com/swinedge/eua-dataset.

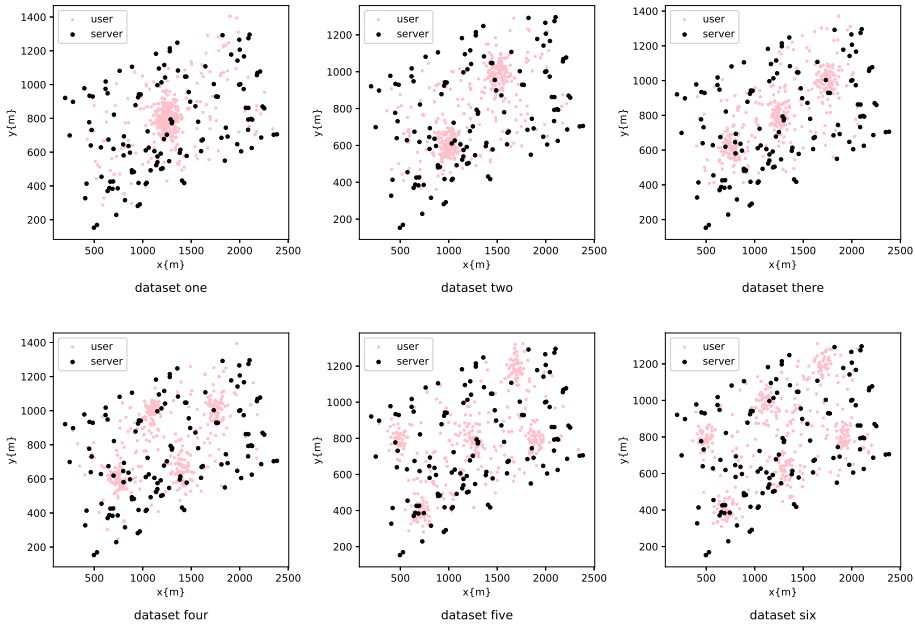


Fig. 2. EUA experimental datasets with different user distributions.

Table 2. Experimental results on different datasets

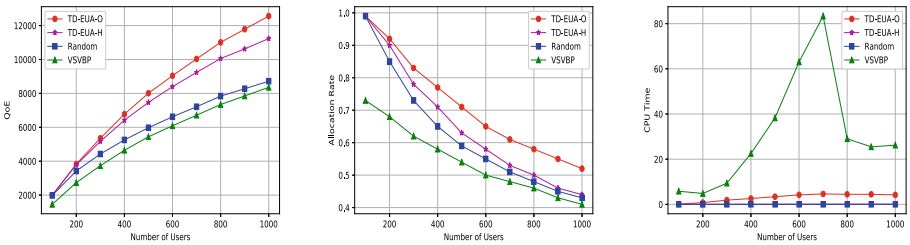
Methods	Dataset one				Dataset two				Dataset three			
	QoE	AR	CPU	Time	QoE	AR	CPU	Time	QoE	AR	CPU	Time
VSVBP	4,479	0.56	22.512		5,319	0.66	24.312		5,189	0.65	23.145	
Random	5,105	0.63	0.014		6,377	0.79	0.016		6,102	0.76	0.015	
TD-EUA-H	6,002	0.66	0.034		7,121	0.82	0.046		6,971	0.80	0.041	
TD-EUA-O	6,357	0.72	2.591		7,416	0.87	3.241		7,261	0.85	2.756	
Methods	Dataset four				Dataset five				Dataset six			
	QoE	AR	CPU	Time	QoE	AR	CPU	Time	QoE	AR	CPU	Time
VSVBP	5,606	0.70	26.235		5,725	0.71	29.324		5,609	0.56	28.165	
Random	6,964	0.87	0.018		7,385	0.92	0.018		6,952	0.86	0.018	
TD-EUA-H	7,500	0.90	0.045		7,665	0.94	0.048		7,507	0.90	0.047	
TD-EUA-O	7,622	0.93	3.212		7,737	0.95	2.234		7,658	0.93	2.946	

5.3 Experimental Results and Analysis

In the experiments, the parameters for existing approaches are tuned to achieve optimal performance. The coverage radius of edge servers obeys a Gaussian distribution with $u = 150$ and the number of tasks per user follows a Gaussian distribution with $u = 3$. Further, we set the number of users to 400, the number of edge servers to 125 and the edge server's available computing capacities follow the Gaussian distribution $N(35, 1)$.

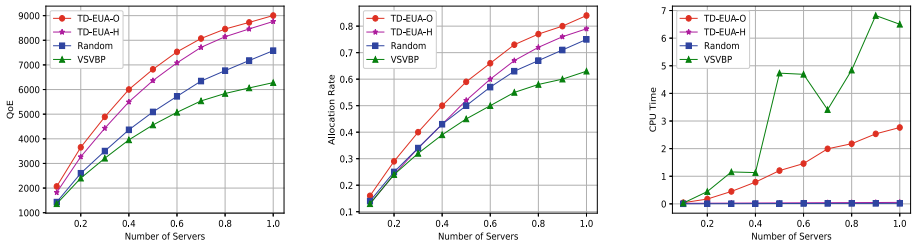
Table 2 summarizes the experimental results, where the best and second-best values in each column are marked in dark and light grey, respectively. The results demonstrate that TD-EUA-O achieves the highest overall QoE and allocates the most users. Specifically, TD-EUA-O outperforms VSVBP, Random and TD-EUA-H by 41.92%, 24.52% and 5.91%, respectively, in QoE. In allocation rate, TD-EUA-O is superior to VSVBP, Random and TD-EUA-H with an advantage of 28.57%, 14.28% and 9.09%, respectively. The main reason of the advantage of TD-EUA-O lies in its consideration of task decomposition and pursuit of global optimization. The computation time of TD-EUA-O is much less than that of VSVBP. Compared to Random and TD-EUA-H, TD-EUA-O takes more time, which is expected because of the NP-hardness of the TD-EUA problem as proved in Sect. 4.2.

Our heuristic approach TD-EUA-H also achieves high performance, with an advantage of 34.00% and 17.57% over VSVBP and Random in QoE, and 17.85% and 4.76% in allocation rate. Surprisingly, we can see that the performance of the Random approach is higher than VSVBP, in terms of both QoE and allocation rate. This is because VSVBP either allocates a user request to an edge server as a whole or does not allocate at all, whereas Random can partition a user request into a set of tasks to be allocated. Overall, the results indicate that decomposing users' requests into tasks can significantly improve the allocation rate and users' overall QoE.



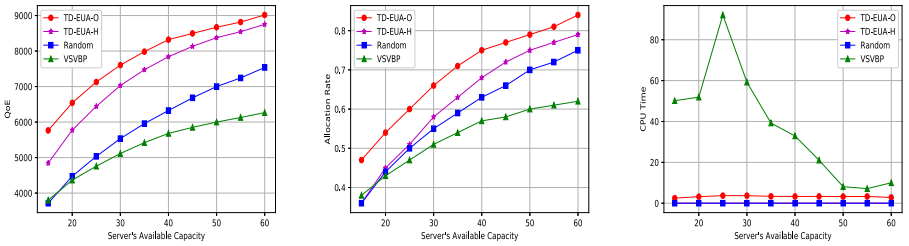
(a) QoE (b) Allocation Rate (c) CPU Time

Fig. 3. Performance comparisons on the variations of edge users.



(a) QoE (b) Allocation Rate (c) CPU Time

Fig. 4. Performance comparisons on the variations of edge servers.



(a) QoE (b) Allocation Rate (c) CPU Time

Fig. 5. Performance comparisons on the variations of server's available capacity.

5.4 Performance Impacts of Parameters

To evaluate the performance of our approaches in various TD-EUA scenarios, we vary the following three parameters in the experiments. Each experiment is repeated 100 times to obtain 100 different user distributions so that the impacts of extreme cases, such as overly sparse or dense distributions, are neutralized.

- **Number of edge users (n):** We random distribute 100, 200, ..., 1,000 edge users in the Melbourne CBD.
- **Number of edge servers (m):** A certain percentage of the all of the edge servers (10%, 20%, ..., 100%) in the Melbourne CBD are included in the experiments.
- **Server's available resources (c):** Edge servers' overall capacity is generated following a Gaussian distribution with $\sigma = 1$. The average capacity of each edge server ranges from $u = 15, 20, \ldots,$ to 60 in each dimension, e.g., CPU, RAM, storage and bandwidth.

Three sets of experiments #1, #2 and #3 are conducted. In each experiment set, we vary one parameter and fix the other two. The results are shown in Figs. 3, 4, and 5.

Figure 3 compares the performance in experiment set #1, where the number of edge users (n) varies from 100 to 1,000 in steps of 100. Figure 3(a) shows

that as n increases, users' overall QoE achieved by different approaches increase. TD-EUA-O achieves the most improvement from 1,989 to 12,559 by 10,570, outperforming TD-EUA-H's 9,244, Random's 6,737, and VSVBP's 6,905. At the beginning, the overall QoE of the four approaches are not that different, because edge servers can provide sufficient computing resources to serve a small number of user requests. As n increases, edge servers' overall computing capacity becomes inadequate, making it hard to achieve high QoE and allocation rate.

The impact of n on allocation rate is shown in Fig. 3(b). Obviously, TD-EUA-O outperforms other approaches again. More specifically, compared with Random and TD-EUA-H, the allocation rate of TD-EUA-O declines more slowly. VSVBP achieves the lowest allocation rate, because it also needs to minimize the number of edge servers hired. Furthermore, as n gradually increases, it is harder to allocate all the users, lowering the allocation rates, from 99.86 % to 52.33 % by 47.33% for TD-EUA-O, from 99.88% to 44.27% by 55.61% for TD-EUA-H, from 99.40% to 43.48% by 55.92% for Random and from 73.01% to 41.96% by 31.04% for VSVBP.

As shown in Fig. 3(c), the average computation time of TD-EUA-O fluctuates slightly and it takes significantly less time consumption than VSVBP. As the problem scales up in n, VSVBP's computation time increases quickly. However, when n exceeds 700, the computation time of VSVBP starts to decrease quickly before it converges. The reason is that the complexity of the TD-EUA problem increases as n increases, producing more possible solutions for VSVBP to inspect. Considering the multi-objective optimization of VSVBP, its solution is not unique, in which it needs to compromise among multiple optimization objectives. Nevertheless, after the turning point, the edge servers cannot accommodate the excessive user requests. Most users are directly allocated to the cloud without further decisions. The computation time of TD-EUA-H is similar to that of Random, slightly less than TD-EUA-O. Thus, TD-EUA-H can accommodate TD-EUA scenarios with large numbers of users.

Figure 4 shows the performance in experiment set #2, where the percentage of the number of edge servers (m) varies from 0.1 to 1.0. As demonstrated, the overall QoE follows a similar trend as in experiment set #1, where TD-EUA-O and TD-EUA-H achieve much higher QoE than Random and VSVBP. As m increases from 10% to 40%, Random achieves performance similar to VSVBP in the resource-scarce situations. Figure 4(b) shows that as m increases, TD-EUA-O and TD-EUA-H continue to achieve high allocation rates. It is worth noting that when m reaches a specific level, the allocation rate achieved by Random is close to that of the TD-EUA-H because the overall computing resources is sufficient to accommodate all the user requests. Figure 4(c) presents the rising trend of the computation time of TD-EUA-O as m keeps increasing. VSVBP takes much more time to find a solution than the other approaches. Its computation time fluctuates. The reason is that VSVBP needs to frequently reselect edge servers to achieve the optimization goal. As for TD-EUA-H and Random, the computation time is always at a low level, similar experiment set #1.

Figure 5 shows the performance comparison in experiment set #3. As the server's available capacity (c) varies from 15 to 60, the overall QoE and allocation rates follow a similar trend as in experiment set #2. In Fig. 5(a), TD-EUA-O and TD-EUA-H outperform the Random and VSVBP in terms of the overall QoE. Especially, With the increase in c, the overall QoE achieved by VSVBP grows slowly, from 3,799 to 6,265 by 2,466, compared with the growth of TD-EUA-O's 3,259 from 5,765 to 9,015, Random's 3,822 from 3,714 to 7,536, and TD-EUA-H's 3,907 from 4,844 to 8,751. Figure 5(b) shows the same trend as Fig. 4(b) on allocation rate as experiment set #2. In Fig. 5(c), compared to other approaches, VSVBP takes the most time to find a solution.

The experimental results show that by considering the task decomposition of service's request, TD-EUA-O and TD-EUA-H outperform the random baseline and the state-of-the-art approach in both QoE and allocation rate with relatively low computation time. In general, TD-EUA-O is the best approach for finding solutions in small-scale instances. In large-scale scenarios, TD-EUA-H is the best option for its second-highest effectiveness of all and its high efficiency.

6 Related Work

With the advances in mobile devices and the Internet of Things, cloud centers may easily be overwhelmed by excessive workloads, causing network latency and congestion. Cisco coined the fog computing, or edge computing, paradigm in 2012 to overcome the major drawback of access latency in cloud computing [1]. Edge computing is an open paradigm that integrates network, computing, storage, and application core capabilities close to end-users to provide low-latency services. Applications deployed and running on the edge can provide fast responses to users' request, meeting their needs for low latency. Service providers can deploy resources on edge servers that are closer to end-users in the edge computing environment. However, an edge server only has a limited computing capacity, making it difficult or sometimes impossible to serve all of the users within its coverage area. Offering many new opportunities, edge computing has also raised a variety of new problems, e.g., edge user allocation (EUA) problem [5,8,10–12,14], edge service placement [2,13,19], edge data management [16–18], edge server placement [3,4], etc.

Recently, the EUA problem as one of the new challenges in the edge computing environment has attracted a lot of attention. Lai et al. [10] made the first attempt to tackle the EUA problem. They modeled the EUA problem as a variable sized vector bin packing problem, and developed an optimal approach for solving the EUA problem with the aim to maximize the number of users allocated and minimize the number of edge servers needed. Then, they further applied user satisfaction as the criterion to measure whether the user allocation is cost-effective, considering that users' resource demands may be differentiated [11]. He et al. [8] proposed a game-theoretic approach for solving the EUA game in a distributed manner. They seek to find the Nash equilibrium of the game as the EUA solution. Peng et al. [14] tackled the EUA problem in an online manner with mobility consideration.

However, existing studies simply assume that a user's demands of computing resources can either be fully fulfilled by a single edge server or cannot be fulfilled at all. In many real-world scenarios, an edge user's request can actually be partitioned into multiple tasks that can be performed by different edge servers. In this paper, we studied the EUA problem with task decomposition and proposed two approaches, TD-EUA-O for finding optimal solutions and TD-EUA-H for finding sub-optimal solutions.

7 Conclusion and Future Work

In this paper, we studied the TD-EUA problem. Instead of serving an user's request as a whole, we consider task decomposition and partition a user's request into individual tasks, which can be performed by different edge servers. To solve the TD-EUA problem, we modeled it as an optimization problem with multiple constraints, and proposed two novel approaches to find TD-EUA solutions that maximize users' overall QoE. The results of experiments conducted on a widely-used real-world dataset demonstrated that our approaches significantly outperform the baseline approach and the state-of-the-art approach. In the future, we will consider the mobility of users and tasks.

Acknowledgments. This work was partially supported by Shanghai Natural Science Foundation (No. 18ZR1414400), National Key Research and Development Program of China (No. 2017YFC0907505), National Natural Science Foundation of China (No. 61772128) and Australian Research Council Discovery Projects (DP18010021 and DP200102491).

References

1. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: International Conference on Mobile Cloud Computing (MCC), pp. 13–16 (2012)
2. Chen, Y., Deng, S., Ma, H., Yin, J.: Deploying data-intensive applications with multiple services components on edge. Mob. Netw. Appl. **25**(2), 426–441 (2020). https://doi.org/10.1007/s11036-019-01245-3
3. Cui, G., He, Q., Chen, F., Jin, H., Yang, Y.: Trading off between user coverage and network robustness for edge server placement. IEEE Trans. Cloud Comput. (2020). https://doi.org/10.1109/TCC.2020.3008440
4. Cui, G., He, Q., Xia, X., Chen, F., Jin, H., Yang, Y.: Robustness-oriented k edge server placement. In: 20th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE (2020). https://doi.org/10.1109/CCGrid49817.2020.00-8
5. Cui, G., et al.: Interference-aware SaaS user allocation game for edge computing. IEEE Trans. Cloud Comput. (2020). https://doi.org/10.1109/TCC.2020.3008440
6. Deng, S., Zhao, H., Fang, W., Yin, J., Dustdar, S., Zomaya, A.Y.: Edge intelligence: the confluence of edge computing and artificial intelligence. CoRR arxiv.org/abs/1909.00560 (2020)

7. Fiedler, M., Hossfeld, T., Tran-Gia, P.: A generic quantitative relationship between quality of experience and quality of service. IEEE Network **24**(2), 36–41 (2010)
8. He, Q., et al.: A game-theoretical approach for user allocation in edge computing environment. IEEE Trans. Parallel Distrib. Syst. **31**(3), 515–529 (2020)
9. Hemmati, M., McCormick, B., Shirmohammadi, S.: QoE-aware bandwidth allocation for video traffic using sigmoidal programming. IEEE MultiMedia **24**(4), 80–90 (2017)
10. Lai, P., et al.: Optimal edge user allocation in edge computing with variable sized vector bin packing. In: Pahl, C., Vukovic, M., Yin, J., Yu, Q. (eds.) ICSOC 2018. LNCS, vol. 11236, pp. 230–245. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03596-9_15
11. Lai, P., et al.: Edge user allocation with dynamic quality of service. In: Yangui, S., Bouassida Rodriguez, I., Drira, K., Tari, Z. (eds.) ICSOC 2019. LNCS, vol. 11895, pp. 86–101. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33702-5_8
12. Lai, P., et al.: Cost-effective app user allocation in an edge computing environment. IEEE Trans. Cloud Comput. (2020). https://doi.org/10.1109/TCC.2020.3001570
13. Li, B., et al.: READ: robustness-oriented edge application deployment in edge computing environment. IEEE Trans. Serv. Comput. (2020). https://doi.org/10.1109/TSC.2020.3015316
14. Peng, Q., et al.: Mobility-aware and migration-enabled online edge user allocation in mobile edge computing. In: IEEE International Conference on Web Services (ICWS), pp. 91–98. IEEE (2019)
15. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
16. Xia, X., et al.: Graph-based optimal data caching in edge computing. In: Yangui, S., Bouassida Rodriguez, I., Drira, K., Tari, Z. (eds.) ICSOC 2019. LNCS, vol. 11895, pp. 477–493. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33702-5_37
17. Xia, X., Chen, F., He, Q., Grundy, J., Abdelrazek, M., Jin, H.: Online collaborative data caching in edge computing. IEEE Trans. Parallel Distrib. Syst. (2020). https://doi.org/10.1109/TPDS.2020.3016344
18. Xia, X., Chen, F., He, Q., Grundy, J., Abdelrazek, M., Jin, H.: Cost-effective app data distribution in edge computing. IEEE Trans. Parallel Distrib. Syst. **32**(1), 31–44 (2021)
19. Xiang, Z., Deng, S., Taheri, J., Zomaya, A.: Dynamical service deployment and replacement in resource-constrained edges. Mob. Netw. Appl. **25**(2), 674–689 (2020). https://doi.org/10.1007/s11036-019-01449-7
20. Zhao, H., Deng, S., Zhang, C., Du, W., He, Q., Yin, J.: A mobility-aware cross-edge computation offloading framework for partitionable applications. In: 2019 IEEE International Conference on Web Services (ICWS), pp. 193–200. IEEE (2019)