

Sharing-Aware Task Offloading of Remote Rendering for Interactive Applications in Mobile Edge Computing

Ruitao Xie, Junhong Fang, Junmei Yao, Xiaohua Jia, *Fellow, IEEE*, and Kaishun Wu

Abstract—Leveraging emerging mobile edge computing and 5G networks, researchers proposed to offload the 3D rendering of interactive applications (e.g. virtual reality and cloud gaming) onto GPU-based edge servers to reduce the user experienced latency. A task offloading problem arises, that is where to offload rendering tasks such that each user will experience tolerable delay and meanwhile the cost of used servers is minimized. The multi-dimensional resource sharing feature of rendering tasks makes the problem challenging. We formulate the task offloading problem into a boolean linear programming. We propose a sharing-aware offloading algorithm which decomposes the problem into two subproblems (user assignment and server packing) and solves them alternately and iteratively. We compare our algorithm with the one without resource sharing in consideration, and the simulations demonstrate that our method can effectively reduce cost as well as satisfy delay requirement.

Index Terms—mobile edge computing, task offloading, delay sensitive, multi-dimensional resource sharing, remote rendering, interactive applications

1 INTRODUCTION

EDGE computing emerges as localized clouds [1], [2]. The telecom carriers are upgrading the infrastructures in existing communication networks to mobile edge computing platforms [3]. There are access sites such as cellular radio base stations, aggregation sites such as those which house distributed antenna systems, and core sites such as central offices. These sites are equipped with computation and storage resources, cooling, power delivery systems *etc.*, and are re-designed to house edge servers. Due to the proximity to users, mobile edge computing can achieve low response time.

Cloud-based interactive applications, such as virtual reality and cloud gaming [4], leverage cloud resources to process their computation-intensive workloads, so as to remove powerful and expensive hardware from user devices and lead to lightweight clients. Pioneering commercial applications include OnLive [5], GeForce Now [6], CloudXR [7] *etc.* However, these interactive applications need high-throughput and low-latency internet connections, which is challenging for users due to the long distance from data centers. A promising solution to overcome this is leveraging emerging mobile edge computing. More specifically, edge-assisted interactive applications offload computation-intensive 3D rendering onto GPU-based infrastructures in mobile edge computing and stream edge-rendered visuals

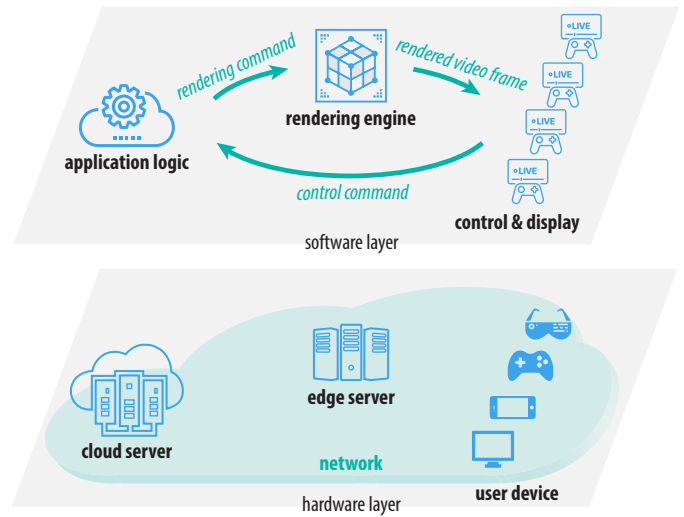


Fig. 1: The system architecture of edge-assisted interactive applications.

to end users over 5G connections, as proposed in [8], [9], as illustrated in Fig. 1. As such, the proximity to end users can greatly reduce latency.

Interactive application service providers lease computation resources from the edge computing platforms to offload rendering tasks. A *task offloading problem* arises, that is where to offload rendering tasks such that every user's delay requirement is satisfied while the cost of used servers is minimized. The cost can be either lease cost or energy consumption. A naive way is to run a rendering task for each user and assign the task to the closest edge node to the user. This can satisfy delay requirement, but may be costly. Because it ignores an important feature of rendering tasks: *multi-dimensional resource sharing*.

Running a rendering task involves several types of resources: storage, memory, GPU memory, CPU, GPU and

- Ruitao Xie, Junhong Fang, Junmei Yao and Kaishun Wu are with the College of Computer Science and Software Engineering, Shenzhen University, China. Email: {drtxie, fangjh111b}@gmail.com, {yaojunmei, wu}@szu.edu.cn.
Kaishun Wu is also with PCL Research Center of Networks and Communications, Peng Cheng Laboratory, Shenzhen, China.
Xiaohua Jia is with the Department of Computer Science, City University of Hong Kong, Hong Kong, China. Email: csjia@cityu.edu.hk.








task/data	shared resources	example							
rendering materials	disk storage	 application							
cached data	main/GPU memory	 instance 1				 instance 2			
rendering operations	CPU/GPU	 rendering task 1	 rendering task 2	 rendering task 3	 rendering task 4				
rendered images	bandwidth	user1	user2	user3	user4	user5	user6	user7	user8

Fig. 2: An illustration of multi-dimensional resource sharing. Eight users participate in two chess games (instances). The users 1-4 in one game and the users 5-8 in another. The rendering tasks for those users are offloaded onto a server. Several types of resources are needed: bandwidth, CPU, GPU, memory, GPU memory and storage. Each user needs a part of bandwidth. Other resources can be shared among multiple users. First, each two users share a view and thus a common rendering task run by CPU and GPU. Second, each two rendering tasks share a common memory (GPU memory as well) since they belong to an instance. Third, all instances share the rendering materials in storage.

bandwidth. Apart from bandwidth, each type of resource can be shared among the users serviced by a server. We take the example as shown in Fig. 2 to illustrate it. First, for an application (a multi-player 3D chess game in our example), the rendering materials (chessboard and pieces) are stored in storage and can be shared among multiple instances. An instance is an execution of the application, a chess game in our example, where each side has two cooperating users. Second, within an instance, some rendering materials are loaded into memory (GPU memory as well) and thus those memory can be shared among all the rendering tasks of the instance. Third, some users may have the same view during the whole process of playing and thus can share a common rendering task run by CPU and GPU, such as user 1 and user 2 in our example. Leveraging this multi-dimensional resource sharing feature, we can save resource consumption by assigning an instance's users together. However, existing related works on the task offloading in edge computing have not considered this, thus are sharing-oblivious. *The main contribution of this paper is to propose a sharing-aware method for the multi-dimensional resource sharing task offloading problem.*

Opposite to the greedy scheme mentioned above, another naive scheme is to maximize possible resource sharing, that is to run a common rendering task for all shared-view users and putting all the tasks belonging to an instance together on a server. However, this may not be able to ensure users' delay requirements, because the users of an instance may spread across the network. Thus, the biggest challenge in our task offloading problem is how to find the best resource sharing plan leading to the least cost while ensuring users' delay requirements.

We formulate the task offloading problem into a boolean linear programming. It is NP-hard in general case. We propose a sharing-aware online offloading algorithm, which alternately and iteratively solves two subproblems: user assignment and server packing. The user assignment subproblem is to assign users towards edge sites, such that each user will experience tolerable delay, and meanwhile resource sharing is maximized. We formulate the user assignment to be the set partition problem. For each site, given a set of users assigned to it, the server packing subproblem is to pack the users into the smallest number of servers

such that resource capacity constraints are satisfied. It can be formulated into the multi-dimensional bin packing problem.

However, resource sharing complicates the server packing subproblem. We may treat each user as an item, a group of shared-view users as an item or all users in an instance as an item, all with multi-dimensional resource demands. These three ways, differing in granularity, result in different packing schemes and cost. Thus, we have to find a level of granularity from three candidates: individual user, shared-view user group, and instance, such that packing cost is minimized. We formulate the granularity decision problem into a three-armed bandit problem, and solve it using a reinforcement learning algorithm.

In addition, we consider one-user-per-instance case particularly, where each instance includes one user only. This corresponds to single-user applications, like single-player games, which are very common. We propose a heuristic offline algorithm based on the transportation problem. We evaluate our algorithms in several scenarios and compare them with a sharing-oblivious offloading algorithm. The simulation results demonstrate that our algorithm can reduce cost significantly.

The rest of this paper is organized as follows. Section 2 introduces the system architecture of edge-assisted interactive applications and the motivation of the rendering task offloading problem. Section 3 presents the problem formulation. Section 4 presents the sharing-aware online offloading algorithm. Section 5 presents the algorithms for the one-user-per-instance case. Section 6 presents simulations and performance evaluations. Section 7 introduces the related works. Section 9 concludes the paper.

2 MOTIVATION

In this section, we introduce the system architecture of edge-assisted interactive applications, and identify the rendering task offloading problem.

2.1 System Architecture

Edge 3D rendering plays an important role in emerging interactive applications, such as AR/VR and video gaming. As shown in Fig. 1, a system may consist of cloud servers, edge servers and user devices. Cloud servers host the core

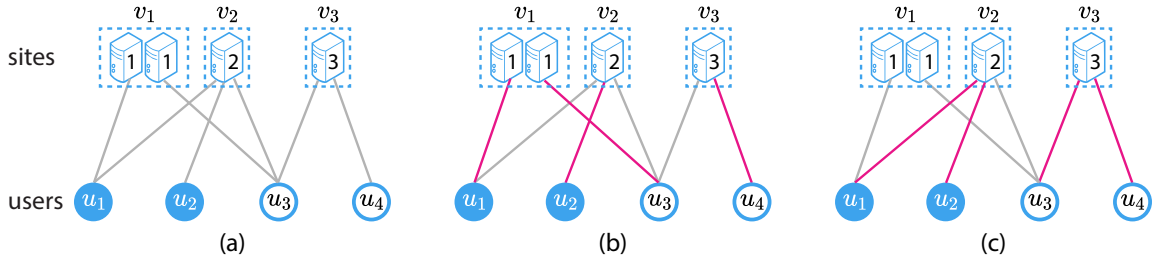


Fig. 3: An example to motivate the optimal task offloading method. An application instance has four users. Users u_1 and u_2 belong to one group; u_3 and u_4 belong to another. Three edge sites $\{v_1, v_2, v_3\}$ contain four servers, each of which has one unit of CPU and unlimited other resources. The number shown on each server icon is the associated cost. The gray edges denote feasible assignments from u to v (considering the delay constraint), and the pink edges denote actual assignments. (a) illustrates the setting of sites and users; (b) illustrates a naive offloading scheme which assigns each user to the least cost site with tolerable delay, and the cost is 7; (c) illustrates the optimal offloading scheme which considers resource sharing, and the cost is 5. (c) is better than (b) because it uses fewer units of CPU, where resource sharing is fully exploited.

logic of an application; edge servers provide rendering and encoding; user devices provide decoding and displaying. An information loop forms among them. Cloud servers generate rendering commands (which manipulates geometric objects to generate video scenes as discussed in [10]) and transfer them to edge servers; edge servers produce video frames and transfer them to user devices; user devices generate control commands (devices' input signals), transfer them backward to cloud servers and let the latter update the application logic.

2.2 Problem & Motivation

An application instance has several users spreading across the network. There are several edge sites, each housing a number of servers. We are about to offload rendering tasks onto some servers in some edge sites. Interactive applications are delay sensitive, thus we have to select offloading edge sites carefully such that every user gets satisfied delay. Besides, the servers in different sites may have distinctive costs. Thus, our objective is to minimize the cost of used servers. In the case of equal cost, we minimize the number of used servers. A naive method is to run a rendering task for each user and assign the task onto the least-cost site with tolerable delay. It may be costly, because it ignores multi-dimensional resource sharing of rendering tasks.

We take an example to illustrate the motivation of our problem. As shown in Fig. 3a, an application instance has four users which belong to two shared-view user groups. Users u_1 and u_2 belong to one group; u_3 and u_4 belong to another. Three edge sites $\{v_1, v_2, v_3\}$ contain four servers, each of which has one unit of CPU and unlimited other resources. In each site, the cost per server is 1, 2 and 3 respectively. We represent the relationship between users and sites as a bipartite graph, where if a user meets delay requirement in a site, then there is an edge between them. As shown in Fig. 3b, the naive method assigns each user onto the least cost site. It uses 4 servers and the total cost is 7. Note that although u_1 and u_3 are both assigned onto the site v_1 , they need two units of CPU because they belong to two different groups and cannot share a common rendering task. In contrast, as shown in Fig. 3c, the optimal solution takes 2 servers and the total cost is only 5. Note that u_1 and u_2 are both assigned onto the site v_2 , they share the same view and thus a common rendering task, which needs one unit of CPU only. Similarly for the users u_3 and u_4 . Thus,

resource sharing can save resource consumption and also cost.

3 PROBLEM FORMULATION

In this section, we first analyze the network delay experienced by a user. Then, we formulate the problem of assigning rendering tasks onto edge servers as a boolean linear programming.

3.1 Delay Analysis

The offloading location of a rendering task affects the delay experienced by a user. As shown in Fig. 1, for an edge-assisted interactive application, the round trip delay experienced by a user consists of the upstream delay of transmitting a control command and the downstream delay of transmitting a rendering command and a rendered video frame. The latter depends on the offloading location of the rendering task associated with the user; while the former does not. Thus, we consider the downstream delay in more detail.

Suppose the rendering task for user u is offloaded onto edge site v , then we analyze its downstream network delay, denoted by t_{uv} . Let r and f denote the average data size of a rendering command and a video frame respectively. Let b_v and d_v denote the downstream throughput and delay from the cloud server to edge site v respectively. Then, we get the delay for transmitting a rendering command is

$$\frac{r}{b_v} + d_v. \quad (1)$$

Let b_{vu} and d_{vu} denote the downstream throughput and delay from edge site v to user device u . Then, similarly we get the delay for transmitting a video frame is

$$\frac{f}{b_{vu}} + d_{vu}. \quad (2)$$

Adding both, we have

$$t_{uv} = \frac{r}{b_v} + \frac{f}{b_{vu}} + d_v + d_{vu}. \quad (3)$$

Thus, we have to carefully offload rendering tasks such that every user gets a satisfied delay.

3.2 Task Offloading Problem

Given an application and a set of instances denoted by I , each instance is an execution of the application program

and is participated by one or several users. On the other hand, there are a set of edge servers denoted by S , which are distributed in several edge sites denoted by V . We assume all edge servers are homogeneous in terms of hardware, but the cost of an edge server depends on the location of its edge site. We are about to offload the rendering tasks of the instances onto the edge servers, such that the total cost is minimized. The following formulation can be easily extended to the servers with heterogeneous hardware, and our algorithms proposed later can be directly used to the heterogeneous servers with various capacities.

We define the task assignment problem here. Given a set of users U and a set of edge servers S , the problem is to assign each user in U to an edge server in S , such that each user's network delay requirement is satisfied, the resource consumption on each edge server is bounded by its capacity, and the total cost of used edge servers is minimized. For each user, a rendering task serving it will run on the assigned server. And on a server, all the shared-view users will share a rendering task.

Let boolean variable x_{uj} denote whether server j hosts a rendering task for user u . Each user has to be assigned onto exactly one server, thus we have

$$\sum_{j \in S} x_{uj} = 1, \quad \forall u \in U. \quad (4)$$

As mentioned earlier, the network delay experienced by a user depends on the offloading site of its task. Let t_{uv} denote the network delay experienced by user u when offloading its task onto site v . For any server j in site v , we have $t_{uj} = t_{uv}$. Then, we get the delay experienced by user u is $\sum_{j \in S} t_{uj} x_{uj}$. It should be bounded by a tolerated maximum value, denoted by τ . Thus, we have

$$\sum_{j \in S} t_{uj} x_{uj} \leq \tau, \quad \forall u \in U. \quad (5)$$

Next we formulate resource constraints. We assume every edge server has enough storage to host the application. Otherwise, some server cannot host any rendering task and are ignored. Let vector $\mathbf{p} = (p^C, p^G, p^M, p^{GM}, p^B)$ denote the capacity of each server for CPU, GPU, memory, GPU memory, and bandwidth respectively. And for simplicity, we assume that a rendering task demands one unit of every type of resource. The multi-dimensional resource sharing feature of rendering tasks complicates the formulation of resource constraints, and thus we have to introduce auxiliary variables.

Suppose the users in U form several shared-view user groups, denoted by K . To formulate the resource constraints for CPU and GPU, we introduce boolean variable y_{kj} denoting whether server j hosts a rendering task for shared-view user group k . Then, the number of tasks on server j becomes $\sum_{k \in K} y_{kj}$. This number may be less than the number of users assigned to the server (i.e., $\sum_{u \in U} x_{uj}$) because of possible task sharing. Let boolean variable w_j denote whether server j is opened. Then, we have capacity constraints:

$$\sum_{k \in K} y_{kj} \leq p^C w_j, \quad \forall j \in S, \quad (6)$$

$$\sum_{k \in K} y_{kj} \leq p^G w_j, \quad \forall j \in S. \quad (7)$$

These constraints can be combined. Let p^K denote the maximum number of tasks allowed by each server, that is

$$p^K = \min \{p^C, p^G\}. \quad (8)$$

We replace the above two constraints by

$$\sum_{k \in K} y_{kj} \leq p^K w_j, \quad \forall j \in S. \quad (9)$$

This constraint ensures that if w_j is zero, all variables y_{kj} are zero.

To formulate the resource constraints for memory and GPU memory, we introduce boolean variable z_{ij} denoting whether server j hosts instance $i \in I$. Then, we have the capacity constraints:

$$\sum_{i \in I} z_{ij} \leq p^M w_j, \quad \forall j \in S, \quad (10)$$

$$\sum_{i \in I} z_{ij} \leq p^{GM} w_j, \quad \forall j \in S. \quad (11)$$

Note that we sum over all instances since any two instances have separate memory and GPU memory. These two constraints can also be combined. Let p^I denote the maximum number of instances allowed by each server, that is

$$p^I = \min \{p^M, p^{GM}\}. \quad (12)$$

Then, we replace the above two constraints by

$$\sum_{i \in I} z_{ij} \leq p^I w_j, \quad \forall j \in S. \quad (13)$$

Different from the above resources which can be shared, we assume bandwidth is not shared among users. Thus, we have the bandwidth constraint

$$\sum_{u \in U} x_{uj} \leq p^B w_j, \quad \forall j \in S. \quad (14)$$

It is noted that the variables introduced above have some relationships. First, x_{uj} and y_{kj} must meet constraints

$$x_{uj} \leq y_{k(u),j}, \quad \forall u \in U, j \in S, \quad (15)$$

where $k(u)$ denotes the shared-view user group to which user u belongs. This means x_{uj} is one only if $y_{k(u),j}$ is one. In other words, the event server j hosts group $k(u)$ is a necessary condition of the event server j hosts user u . A similar relationship exists between y_{kj} and z_{ij} . We know if server j hosts group k , then it must host the instance of this group. Let $i(k)$ denote the instance which includes group k , then we have

$$y_{kj} \leq z_{i(k),j}, \quad \forall k \in K, j \in S. \quad (16)$$

At last, the necessary condition $z_{ij} \leq w_j$ has been suggested by (13).

Our objective is to minimize the total cost which the application provider will pay to an edge computing service provider. Let c_v denote the cost per server in site v , then we have $c_j = c_v$, for any server j in site v . Our objective is to minimize $\sum_{j \in S} c_j w_j$. Thus, the problem is formulated into a boolean linear programming as follows,

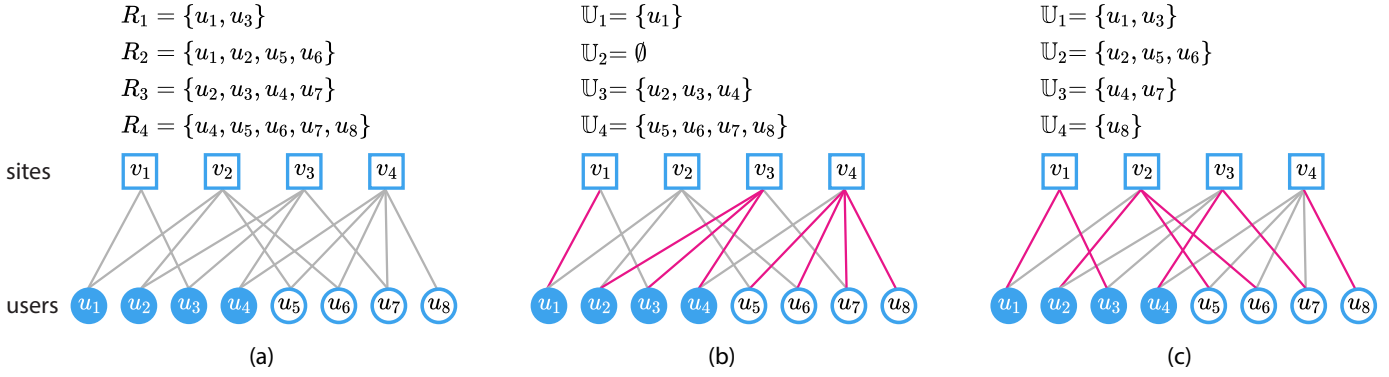


Fig. 4: The bipartite graph includes a set of four sites (squares) and a set of eight users (circles). Solid and empty circles differentiate two shared-view user groups. The gray edges denote feasible assignments from a user to a site (considering the delay constraint), and the pink edges denote actual assignments. (a) illustrates the set of users satisfying delay requirements in each site, that is R_v ; (b) illustrates an assignment scheme $\{\{u_1\}, \emptyset, \{u_2, u_3, u_4\}, \{u_5, u_6, u_7, u_8\}\}$; (c) illustrates another assignment scheme $\{\{u_1, u_3\}, \{u_2, u_5, u_6\}, \{u_4, u_7\}, \{u_8\}\}$.

$$\begin{aligned} \min \quad & \sum_{j \in S} c_j w_j \\ \text{s.t.} \quad & \sum_{j \in S} x_{uj} = 1, \quad \forall u \in U \\ & \sum_{j \in S} t_{uj} x_{uj} \leq \tau, \quad \forall u \in U \\ & \sum_{u \in U} x_{uj} \leq p^b w_j, \quad \forall j \in S \\ & \sum_{k \in K} y_{kj} \leq p^k w_j, \quad \forall j \in S \\ & \sum_{i \in I} z_{ij} \leq p^i w_j, \quad \forall j \in S \\ & x_{uj} \leq y_{k(u),j}, \quad \forall u \in U, \quad j \in S \\ & y_{kj} \leq z_{i(k),j}, \quad \forall k \in K, \quad j \in S \\ & x_{uj} \in \{0, 1\} \quad \forall u \in U, \quad j \in S \\ & y_{kj} \in \{0, 1\} \quad \forall k \in K, \quad j \in S \\ & z_{ij} \in \{0, 1\} \quad \forall i \in I, \quad j \in S \\ & w_j \in \{0, 1\} \quad \forall j \in S \end{aligned} \quad (17)$$

4 SHARING-AWARE OFFLOADING ALGORITHM

Application instances arrive sequentially, thus we propose a sharing-aware online offloading algorithm to solve the problem (17). In other words, our algorithm deals with instances in the order of their arrivals.

The essence of our algorithm is to decompose the problem into two subproblems: user assignment and server packing. Our heuristic algorithm solves these two subproblems alternately and iteratively. The user assignment problem is to assign the users to the edge sites such that each user can receive satisfied delay at the assigned site and resource sharing is maximized. After user assignment, we define a server packing problem for each site. It is to pack the users having been assigned to the given site into a set of servers such that resource capacity constraints are satisfied and the number of used servers is minimized.

However, an issue arises here. In this decomposition, we do not consider the number of servers allowed in each site in the user assignment. Thus, a site may not be able to accommodate all the users assigned to it. In other words, the required number of servers to pack those users may exceed the maximum value allowed. In this situation, some

instances will be reassigned later. In following, we formulate the subproblems and introduce our algorithm.

4.1 User Assignment Problem & Algorithm

The user assignment problem is to assign the users to the edge sites such that each user receives tolerable delay and resource sharing is maximized. Eventually, the given set of users are divided into disjoint subsets, each being assigned to a site. Different divisions lead to different resource consumption, thus different costs. To find the best division is a set partition problem.

We take the example in Fig. 4 to illustrate it. Given a set of 4 edge sites (illustrated by squares) and a set of 8 users (illustrated by circles). The users belong to an instance, but 2 shared-view user groups, which are differentiated by solid and empty circles. We represent the relationship between users and sites as a bipartite graph, that is if a user meets delay requirement in a site, then there is an edge between them. Each site thus has a set of users connecting to it, denoted by R_v for site v . For example in Fig. 4a, $R_3 = \{u_2, u_3, u_4, u_7\}$. The users assigned onto the site v must satisfy delay requirement, thus must be a subset of R_v . Moreover, each user must be assigned onto exactly one site. Thus, the sets of users assigned to each site must be pair-wise disjoint, and the union of them equals to the whole set of users. In other words, the result of user assignment forms a set partition. For example, the user assignment scheme $\{\{u_1\}, \emptyset, \{u_2, u_3, u_4\}, \{u_5, u_6, u_7, u_8\}\}$ in Fig. 4b is a set partition. The user assignment scheme $\{\{u_1, u_3\}, \{u_2, u_5, u_6\}, \{u_4, u_7\}, \{u_8\}\}$ in Fig. 4c is another set partition. Both assignment schemes are valid, but they may lead to different offloading costs. The assignment scheme in Fig. 4b needs 3 rendering tasks, each running in sites v_1, v_3 and v_4 respectively, because in the site v_3 the users u_2, u_3 and u_4 share a view and thus a common rendering task. It is similar for the users assigned to site v_4 . In contrast, the assignment scheme in Fig. 4c needs 1, 2, 2 and 1 rendering tasks in each site, 6 in total. More tasks lead to higher cost.

To tackle this problem, we first give a formal definition, and then present a heuristic algorithm.

4.1.1 Formal Definition

Let U denote a set of users for an instance, which differentiates the notation U used in the previous section denoting

the set of users of all instances. Given \mathbb{U} and a set of edge sites V , the user assignment problem is to divide \mathbb{U} into a partition. A partition is an indexed family of subsets of \mathbb{U} with index set V , denoted by $\{\mathbb{U}_v : v \in V\}$, which satisfies two constraints: 1) the union of those subsets equals to \mathbb{U} , that is

$$\bigcup_{v \in V} \mathbb{U}_v = \mathbb{U}; \quad (18)$$

2) those subsets are pair-wise disjoint, that is

$$\mathbb{U}_{v_1} \cap \mathbb{U}_{v_2} = \emptyset, \quad \forall v_1, v_2 \in V, v_1 \neq v_2. \quad (19)$$

Partition part \mathbb{U}_v corresponds to the users assigned to site v . Partitioning ensures that each user is assigned to exact one edge site.

The user assignment has to ensure that every user gets satisfied delay at the assigned site. Let R_v denote all the users in \mathbb{U} satisfying delay requirement in site v , then we have

$$R_v = \{u \in \mathbb{U} \mid t_{uv} \leq \tau\}. \quad (20)$$

Thus, partition part \mathbb{U}_v for edge site v must satisfy

$$\mathbb{U}_v \subseteq R_v. \quad (21)$$

In order to bring this constraint into the set partition problem formulation, we introduce a tuple $\langle v, A \rangle$ denoting site v and a feasible set on this site, that is $A \subseteq R_v$. Then, we have a family of candidate tuples

$$\mathcal{A} = \{\langle v, A \rangle : v \in V, A \subseteq R_v\}. \quad (22)$$

Thus, our problem becomes given \mathbb{U} and \mathcal{A} , to select a subset of \mathcal{A} such that it forms a partition of \mathbb{U} , which means the second elements of our selected tuples satisfy constraints (18) and (19).

We can get many partitions of \mathbb{U} from \mathcal{A} . However, different partitions lead to different kinds of resource sharing and thus different resource (memory and processor) consumption. For example, considering a partition where each user is assigned to different edge sites, then there is no resource sharing and the resource consumption is high. In contrast, considering another partition where all users are assigned to the same edge site, then resource can be shared if they are packed in the same server, so we say the possibility of sharing resource is high and it may lead to low memory consumption.

Given a family of candidate tuples \mathcal{A} defined in (22), we prefer to choose the set with large cardinality, since it brings high possibility of resource savings. Besides, we prefer the edge site with low cost. Combining them together, we define the cost for tuple $\langle v, A \rangle \in \mathcal{A}$ as

$$c(\langle v, A \rangle) = c_v |A|^{-\theta}. \quad (23)$$

The lower the better. θ is the weight of $|A|$, and balances the server cost and set cardinality, which affects the extent of resource sharing.

Thus, we define the **user assignment (UA) problem** for an instance as: given a set of users \mathbb{U} of an instance, a family of candidate tuples \mathcal{A} defined in (22) and a cost function $c : \mathcal{A} \rightarrow \mathbf{R}^+$, find a minimum cost subset of \mathcal{A} such that it forms a partition of \mathbb{U} . This is a set partition problem. This formulation ensures that every user is assigned to exact one edge site and it can get satisfied delay at that site. As such, the first two constraints in (17) can be met.

4.1.2 Heuristic Algorithm

The set partition problem is NP-hard. It is inefficient to directly solve it, because \mathcal{A} is exponential size. We propose an efficient heuristic algorithm to solve it, as shown in Algorithm 1. Instead of choosing a partition from the exponential-size set \mathcal{A} , we start with a subset of it

$$\mathcal{A}' = \{\langle v, R_v \rangle : v \in V\}, \quad (24)$$

which only has $|V|$ elements in total. Our algorithm iteratively selects partition parts from \mathcal{A}' . In each iteration, it selects the tuple whose cost $\frac{c(\langle v, A \rangle)}{|A|}$ is the smallest (say $\langle v^*, A^* \rangle$) as a partition part (line 4-5). Next, to ensure that the chosen partition parts are pair-wise disjoint, we update set \mathcal{A}' by removing the users just assigned (*i.e.*, A^*) from each element (line 6-7). Finally, we remove invalid elements $\langle v, \emptyset \rangle$ from set \mathcal{A}' (line 8-9). The algorithm iterates until the union of the chosen partition parts equals to \mathbb{U} (line 3).

Algorithm 1 User Assignment Algorithm (UA)

Input: \mathbb{U} and V

Output: partition $\{\mathbb{U}_v : v \in V\}$

- 1: $\mathcal{A}' = \{\langle v, R_v \rangle : v \in V\}$
 - 2: $\mathbb{U}_v \leftarrow \emptyset, \quad \forall v \in V$
 - 3: **while** $\bigcup_{v \in V} \mathbb{U}_v \neq \mathbb{U}$ **do**
 - 4: From \mathcal{A}' find the tuple whose cost-effectiveness $\frac{c(\langle v, A \rangle)}{|A|}$ is the smallest, say $\langle v^*, A^* \rangle$
 - 5: $\mathbb{U}_{v^*} \leftarrow \mathbb{U}_{v^*} \cup A^*$
 - 6: **for** $\langle v, A \rangle \in \mathcal{A}'$ **do**
 - 7: $A \leftarrow A \setminus A^*$
 - 8: **if** $A = \emptyset$ **then**
 - 9: Remove $\langle v, A \rangle$ from \mathcal{A}'
-

4.2 Server Packing Problem & Algorithm

Once the users have been assigned to the edge sites, we address server packing. For each site, given a set of users assigned to it, we pack them into the smallest number of servers. As formulated in problem (17), each server has three kinds of resource capacity constraints, which are the limitations on the number of instances, shared-view user groups and users respectively. Thus, the server packing can be formulated into a three-dimensional bin packing problem.

However, resource sharing complicates the server packing, since collocating some users together may reduce resource consumption. We can treat each user as an item with resource demand $\langle 1, 1, 1 \rangle$ to pack in a bin with capacity $\langle p^I, p^K, p^B \rangle$. Here each element in these tuples corresponds to the required (or maximum) number of instances, tasks, and users for an item (or bin) respectively. Alternatively, we may collocate the users belonging to a shared-view user group together and treat it as an item with resource demand $\langle 1, 1, n^B \rangle$, where n^B is the number of users collocated in this group. Similarly, we may collocate the users in an instance together and treat it as an item with resource demand $\langle 1, n^K, n^B \rangle$, where n^K and n^B are the number of tasks (*i.e.*, user groups) and users collocated in this instance respectively. Different levels of granularity may lead to different costs. The coarse granularity (instance) maximizes resource sharing, but may cause resource waste due to large

item size. In contrast, the fine granularity (user) does not maximize resource sharing, but may use resource more sufficiently due to small item size. We have to find the best level of granularity such that packing cost is minimized.

From our simulations, we find that the best level of granularity depends on several factors, such as the server capacity $\langle p^l, p^k, p^b \rangle$, the delay limit τ , and whether the server costs are equal over edge sites or not and so on. It is hard to get a decision formula via an analytical model. Instead, we propose to use reinforcement learning method to learn the best granularity from experiences in any situation.

We formulate the granularity decision problem into a three-armed bandit problem. Given three actions (levels of granularity), at each step, the system chooses an action, offloads k instances successively, and does server packing with this action. A number of servers will be opened to pack those instances. The fewer the better. Thus, we define the reward (denoted by R) for an action as the minus of the number of newly opened servers (denoted by C), that is $R = -C$. Our objective is to maximize the expected total reward over a long term of action selections.

We solve this problem using a simple action-value approach [11] as shown in Algorithm 2. Let $Q(a)$ denote the estimated value function of action a . We use it to evaluate the goodness of selecting each action. We get $Q(a)$ by finding the steps of taking action a and calculating the average of the obtained rewards in these steps. The initial values of $Q(a)$ highly affects the convergence. We initialize $Q(a)$ by averaging the received rewards of taking action a over m steps (line 1). Let $N(a)$ denote the selected count of action a , and we initialize it to m (line 2). After initialization, at each step, we select the action with the highest estimated value with probability of $1 - \xi$ and select the action randomly with the probability of ξ (line 4). Then, we update the value function for the chosen action by averaging the received rewards (line 5-7).

Algorithm 2 Granularity Decision Algorithm (GD)

- 1: $Q(a) \leftarrow$ average reward over m steps, $\forall a$
 - 2: $N(a) \leftarrow m$, $\forall a$
 - 3: **loop forever**
 - 4: $a' \leftarrow \begin{cases} \arg \max Q(a) & \text{with probability } 1 - \xi \\ \text{a random action} & \text{with probability } \xi \end{cases}$
 - 5: $R \leftarrow$ -the number of newly opened servers
 - 6: $N(a') \leftarrow N(a') + 1$
 - 7: $Q(a') \leftarrow Q(a') + [R - Q(a')]/N(a')$
-

In the above algorithm, the parameter ξ balances exploration and exploitation. The parameter m affects the quality of initialization and further influences the learning process. The larger m is, the closer $Q(a)$ is to its true value. But large m may defer convergence. In addition, the parameter k is the number of instances to process in each step. It affects the quality of reward. If it is too small, extremely as one, the reward samples for every action are very similar or even zero so that they cannot differentiate actions. On the other hand, if it is too large, lots of instances are required for the update of value functions and also slow down convergence.

With a level of granularity, we group the users into items and pack them into the smallest number of servers such that the three-dimensional resource capacity is satisfied. The

three-dimensional bin packing problem is NP-hard. There are many approximation or heuristic algorithms to approach it. We use the traditional first-fit algorithm. When packing a new item, we scan the opened servers in order and place the item in the first server that fits. If it does not fit in any of the existing servers, we open a new one. To determine whether an item fits a server, we have to evaluate the resource capacity constraints (9), (13) and (14) with resource sharing considered as listed in the formulation (17). It is noted that our granularity decision algorithm works with any bin packing algorithm.

We look at a special case of our problem, where there is only one user group in each application instance. In this situation, user group and instance are one-to-one mapping. The two variables in our formulation (17), y_{kj} and z_{ij} , become the same, thus the two corresponding capacity constraints degenerate to the following one

$$\sum_{i \in I} z_{ij} \leq \min\{p^k, p^l\} w_j, \quad \forall j \in S. \quad (25)$$

As a result, our server packing problem becomes the cardinality constrained bin packing problem, which is still NP-hard. Several approximation algorithms have been proposed to solve it. For example, Babel *et al.* in [12] proposed an approximation algorithm with an asymptotic worst-case performance ratio of 2.

4.3 Online Offloading Algorithm

Next, we introduce our online offloading algorithm based on the components introduced so far. Our algorithm offloads instances one by one in the order of their arrivals. For each instance, we solve the user assignment problem first and then solve the server packing problem for each site later. However, a site may not be possible to accommodate all the users assigned to it, as we do not consider the number of servers allowed in each site during user assignment. Thus, we do server packing for sites one by one in the nondecreasing order of cost. The idea is to fill up the least cost site first. In the case of a tie, the site with more servers takes precedence in order to avoid overflow in server packing.

Algorithm 3 illustrates the sharing-aware online offloading algorithm for any instance \mathbb{U} . It proceeds as follows. We sort the sites in the nondecreasing order of cost and will do server packing iteratively in this order until all the users are packed (line 1). If there is a tie, the site with more servers has higher priority. In each iteration, we deal with the current least cost site l . First, we run the user assignment (UA) algorithm for the remaining users \mathbb{U} and the remaining edge sites V' , and get partition $\{\mathbb{U}_v : v \in V'\}$ (line 4-5). Second, we get the best granularity from GD algorithm (line 6), and do server packing using the first-fit algorithm until running out of servers (line 7). Third, we remove the packed users from \mathbb{U} (line 8). Fourth, we try the next possible site until either all users have been packed or all sites have been tried out (line 3).

After the algorithm terminates, if there are users remained in \mathbb{U} , then these users are unable to host due to the scarcity of resources. Our system can admit the packed users and reject the remaining ones. Alternately, the system can

reject the entire instance. This is determined by the agreement between the edge service provider and the application provider.

We analyze that our algorithm has polynomial complexity given polynomial-time algorithms for the **UA** problem and the **SP** problem. Given the number of sites n_v , we get the worst-case complexity of our algorithm is

$$O(n_v \log n_v) + O(n_v T_{\text{UA}}) + O(n_v T_{\text{SP}}), \quad (26)$$

where T_{UA} and T_{SP} denote the time complexity of the **UA** algorithm and the **SP** algorithm respectively. The first term is for sorting the sites. The second term is for solving the user assignment problems, n_v at most. The third term is for solving the server packing problems, n_v at most. Our user assignment algorithm has $O(n_v^2)$ complexity. The first-fit algorithm used for the **SP** problem has a constant time complexity, since in each site the number of items to pack is bounded by the number of users per instance, which is a constant. It is noted that the granularity decision algorithm also has $O(1)$ time complexity.

Algorithm 3 Sharing-Aware Offloading Algorithm (SAO)

Input: \mathbb{U} and V

Output: Assign each user in \mathbb{U} to a server

- 1: Sort V in the nondecreasing order of cost
 - 2: $l \leftarrow 1$
 - 3: **while** $\mathbb{U} \neq \emptyset$ and $l \leq |V|$ **do**
 - 4: $V' \leftarrow \{l, l+1, \dots, |V|\}$
 - 5: $\{\mathbb{U}_v : v \in V'\} \leftarrow \text{UA}(\mathbb{U}, V')$
 - 6: Get a level of granularity from **GD**
 - 7: Pack \mathbb{U}_l using the first-fit algorithm in the selected granularity until running out of servers
 - 8: Remove the packed users from \mathbb{U}
 - 9: $l \leftarrow l+1$
-

It is noted that we run a single granularity decision algorithm for all sites in SAO. In this way, we collect the number of newly opened servers from sites, sum up and get the reward. In contrast, another way to implement is to run a GD algorithm for each site. The latter has an advantage over the former when sites have heterogeneous servers and thus different choices of granularity. In our simulation later, we use the first implementation due to the homogeneous assumption.

The above sharing-aware offloading algorithm can handle the heterogeneous servers with various capacities and also dynamic instance arrivals and departures. Because in the entire algorithm, we use server capacity information only in the step of server packing (line 7), where the latest remaining resource capacity is considered.

5 ONE-USER-PER-INSTANCE CASE

In this section, we look at a special case of our problem, where there is only one user in each application instance. This situation is also very common in practice. Note that in this situation, only the rendering materials in storage can be shared among users, not for memory and processor. We will propose new methods to approach it. In this situation, user, user group and instance are all one-to-one mapping. The three variables in our formulation (17), x_{uj} , y_{kj} and z_{ij} ,

become the same. Moreover, users are homogeneous so that server packing is trivial. As such, we can reformulate the problem in a simple way as follows,

$$\begin{aligned} \mathbf{P:} \quad & \min \quad \sum_{v \in V} c_v \left\lceil \frac{\sum_{u \in U} x_{uv}}{p^u} \right\rceil \\ & \text{s.t.} \quad \sum_{v \in V} x_{uv} = 1, \quad \forall u \in U \\ & \quad \sum_{v \in V} t_{uv} x_{uv} \leq \tau, \quad \forall u \in U \\ & \quad \sum_{u \in U} x_{uv} \leq p^u \hat{m}_v, \quad \forall v \in V \\ & \quad x_{uv} \in \{0, 1\} \quad \forall u \in U, v \in V. \end{aligned} \quad (27)$$

We explain this formulation in detail. Parameter p^u denotes server capacity and equals $\min\{p^k, p^l, p^b\}$. It is an integer. In other words, it represents the maximum number of users allowed on each server. Our variable x_{uv} denotes whether assigning user u to site v . The objective still is to minimize the cost of used servers. Here, the term $\left\lceil \frac{\sum_{u \in U} x_{uv}}{p^u} \right\rceil$ is the number of the servers consumed in site v , by rounding up the ratio of the number of users assigned onto site v to the server capacity. The first two constraints are similar to that in (17). The third constraint ensures the site capacity is satisfied, that is the number of users assigned to site v cannot exceed the maximum number $p^u \hat{m}_v$, where \hat{m}_v is the maximum number of servers in site v .

5.1 Offline Algorithm

We propose an offline heuristic algorithm to solve the problem (27). We first relax the original objective function in (27) and get the following approximation problem,

$$\begin{aligned} \mathbf{AP:} \quad & \min \quad \sum_{v \in V} c_v \frac{\sum_{u \in U} x_{uv}}{p^u} \\ & \text{s.t.} \quad \text{constraints in (27)}. \end{aligned} \quad (28)$$

Then, this problem can be equivalently transformed to a classical **transportation problem** where the objective is to minimize the cost of distributing a product from a set of sources to a set of destinations. Here, sites are sources, users are destinations, and the resource requirement of a user is a product.

We transform the above problem (28) to a standard transportation problem by removing its delay constraints. The way is to define a new cost as below,

$$c'_{uv} = \begin{cases} \frac{c_v}{p^u} & \text{if } u \text{ gets satisfied delay in site } v \\ +\infty & \text{otherwise.} \end{cases} \quad (29)$$

As such, the problem (28) is equivalently transformed to the following transportation problem.

$$\begin{aligned} \mathbf{TP:} \quad & \min \quad \sum_{v \in V} \sum_{u \in U} c'_{uv} x_{uv} \\ & \text{s.t.} \quad \sum_{v \in V} x_{uv} = 1, \quad \forall u \in U \\ & \quad \sum_{u \in U} x_{uv} \leq p^u \hat{m}_v, \quad \forall v \in V \\ & \quad x_{uv} \in \{0, 1\} \quad \forall u \in U, v \in V. \end{aligned} \quad (30)$$

If a feasible solution to the above problem has an infinite objective value, then it implies that some user is assigned to some site with unsatisfactory delay. Otherwise, the delay constraint in the **P** problem is satisfied, and thus the solution is also feasible for the **P** problem. It is noted that in the above problem the boolean constraints are unnecessary any more, since the theory on linear programming have demonstrated that if the values $p^U \hat{m}_v$ are integers (which is true in practice), then the optimal solution produced by the simplex algorithm also are integers [13]. There are exact solvers for the transportation problem, such as transportation simplex method, and some heuristics such as the northwest corner method and Vogel's method [14].

Algorithm 4 Transportation-based Offline Offloading (TFO)

Input: U and V

Output: Assign each user in U to a site

- 1: Initialize $m_v \leftarrow \hat{m}_v, \forall v \in V$
- 2: **while** $\text{TP}(\{m_v, \forall v \in V\})$ is feasible and has a finite objective value **do**
- 3: Solve $\text{TP}(\{m_v, \forall v \in V\})$ and get a feasible solution
- 4: $\{x_{uv}, \forall u \in U, v \in V\}$ with a finite objective value
- 5: $h_v \leftarrow \left\lceil \frac{\sum_{u \in U} x_{uv}}{p^v} \right\rceil, \forall v \in V$
- 6: $v^* \leftarrow \arg \max_{v \in V} h_v - \frac{\sum_{u \in U} x_{uv}}{p^v}$
- 7: $m_v \leftarrow h_v, \forall v \in V$
- 8: $m_{v^*} \leftarrow m_{v^*} - 1$

We propose a transportation-based offline offloading (TFO) algorithm as shown in Algorithm 4. It iteratively solves a series of the relaxed transportation problems (**TP** for short) in (30) with different site capacities. We define the capacity of a site as the maximum number of servers allowed to use, denoted by m_v and initialized as \hat{m}_v for site v . Our algorithm iteratively shrinks the site capacities until either the **TP** problem is infeasible or it does not have a finite objective value (line 2). In each iteration, we first solve the **TP** problem with the site capacity of m_v , denoted by $\text{TP}(\{m_v, \forall v \in V\})$, get a feasible solution which leads to a finite objective value, denoted by $\{x_{uv}, \forall u \in U, v \in V\}$ (line 3-4). According to the aforementioned discussion, the solution is also feasible to the **P** problem. Then, we find the site with the largest spare server capacity (line 5-6), which is the spare capacity of the server not fully filled, say v^* . It equals to $h_v - \frac{\sum_{u \in U} x_{uv}}{p^v}$, where h_v is the number of servers used in site v . Next, we update the capacity of each site to h_v and reduce the capacity of site v^* by one (line 7-8).

5.2 Online Algorithm

For this special case, the proposed SAO algorithm degenerates to a simple one. It assigns each user to the least-cost site with tolerable delay. Each instance has a single user, thus the server packing algorithm in SAO degenerates to the first-fit algorithm in the granularity of user.

6 PERFORMANCE EVALUATION

Our sharing-aware offloading algorithm (SAO) considers resource sharing in offloading rendering tasks. We compare

TABLE 1: Parameter setting for the simulation environment. $\text{Unif}\{a, b\}$ denotes discrete uniform distribution between a and b .

Parameter	Value
The number of groups per instance	2
The number of users per group	4
The number of sites	50
The number of servers per site	$\text{Unif}\{50, 100\}$
Server cost in each edge site (c_v)	$\text{Unif}\{1, 10\}$
$\langle p^1, p^k, p^B \rangle$	$\langle 5, 10, 20 \rangle$
Delay t_{uv} (ms)	$\text{Unif}\{10, 50\}$
Delay limit τ (ms)	30

it with an algorithm without resource sharing in consideration, called sharing-oblivious offloading algorithm (SBO). It is also an online algorithm, and solves the user assignment subproblem and the server packing subproblem alternately and iteratively as in our algorithm. But it uses different user assignment algorithm and server packing algorithm. Specifically, for each instance, the SBO assigns each user to the least-cost edge site with tolerable delay. In the case of a tie, the site with the most servers is selected. Besides, the SBO implements server packing using the first-fit algorithm in the granularity of user. It is noted that in the first-fit algorithm to determine whether an item fits a server, we have to evaluate the resource capacity constraints (9), (13) and (14) with resource sharing considered as listed in the formulation (17).

In addition, to demonstrate the advantage of our algorithm, we also compare it with packing in the granularity of user (SAO-U), user group (SAO-G), and instance (SAO-I) in the following evaluation. We use the cost reduction ratio defined below to evaluate performance,

$$\text{cost reduction ratio} = \frac{\text{cost}(\text{SBO}) - \text{cost}(\text{SAO})}{\text{cost}(\text{SBO})}. \quad (31)$$

In our simulation, if not stated otherwise, we set the parameters as presented in Table 1. We vary the number of instances to allocate. These instances start sequentially and keep running until the end of the simulation. Each result is averaged over 20 random simulations. We set the parameters for our algorithm as follows: θ is 1, ξ is 0 for full exploitation, k is 200 and m is 1. As such, in the granularity decision algorithm, the initialization of value functions takes $3mk = 600$ instances. Moreover, in the GD algorithm, we start initialization from the second step, because in the first step packing is from scratch and this may lead to a high-bias reward.

6.1 Reduction of Cost

We first show that our algorithm can reduce cost significantly compared with the SBO algorithm. As illustrated in Fig. 5, our algorithm (SAO) reduces cost by up to 52%. The performance improvement comes from two aspects: 1) the sharing-aware user assignment algorithm; 2) the granularity decision algorithm. First, it is observed that SAO-U gets 50% cost reduction compared to SBO. This suggests leveraging resource sharing leads to cost reduction. Second, it is observed that SAO is better than SAO-U by around two percentage points. This suggests our granularity decision algorithm can further reduce cost.

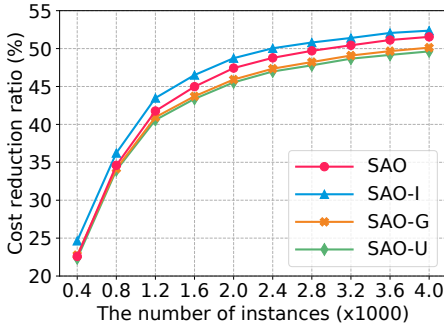


Fig. 5: The cost reduction ratio as the number of instances increases.

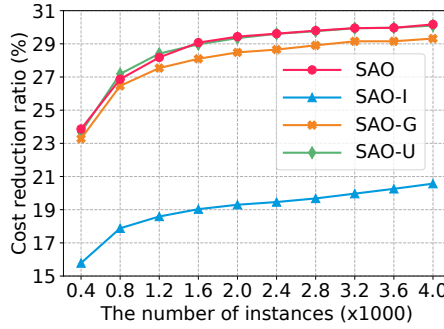


Fig. 6: The cost reduction ratio under equal server cost.

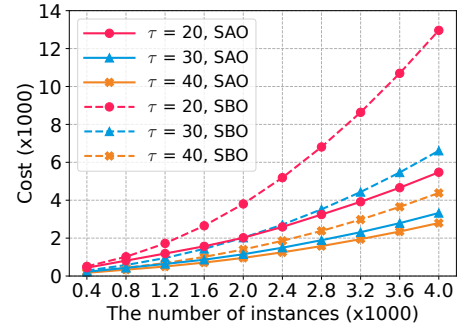


Fig. 7: The cost under various delay limit (τ) values.

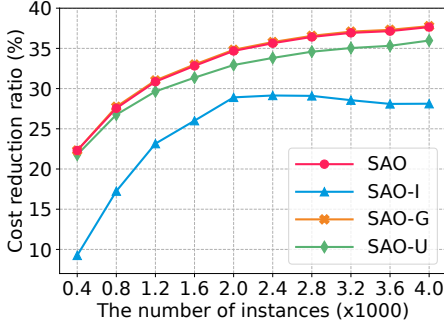


Fig. 8: The cost reduction ratio under the delay limit of 40 ms.

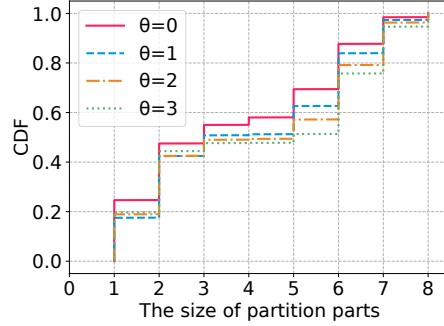


Fig. 9: The empirical CDF for the size of partition parts under various θ values.

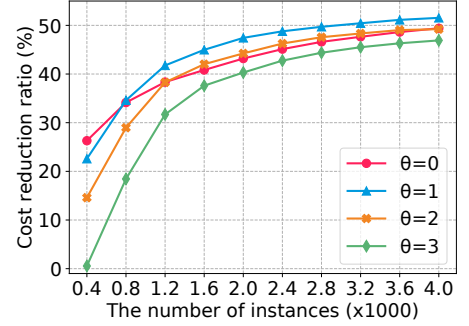


Fig. 10: The cost reduction ratio of SAO under various θ values.

TABLE 2: The convergence results of the GD algorithm under various situations with 4000 instances over 20 simulations.

Environment parameters	Instance	Group	User
Default τ , default server cost	16	4	0
Default τ , equal server cost	0	4	16
$\tau = 40$, default server cost	0	20	0

As listed in the first line in Table 2, with 4000 instances, over 20 random simulations, the granularity decision algorithm converges to the level of instance in 16 cases and to the level of group in 4 cases. This performance can be further improved by starting with better initial value functions, which can be estimated with large m and k , and learning with more instances.

Next, we set equal server cost. As shown in Fig. 6, our algorithm can reduce cost by up to 30%. In this situation, SAO-I performs severely poorly. This is because in the case of equal cost, $|A|^{-\theta}$ dominates the cost in the definition (23) and as a result the users in an instance aggregate tightly. Packing in the granularity of instance causes resource waste due to large item size, although resource sharing is maximized. In contrast, SAO-U performs best, because small item size leads to sufficient resource usage. SAO nearly learns the best performance. As listed in the second line in Table 2, with 4000 instances, over 20 random simulations, the GD algorithm converges to the level of group in 4 cases and to the level of user in 16 cases. It successfully avoids the level of instance and nearly converges to the best level.

6.2 Effect of τ

In this section, we evaluate how the delay limit affects performance. We compare the cost under three values of τ : 20 ms, 30 ms, and 40 ms. Since the SBO algorithm needs more servers when τ is 20 than that when τ is 30, we set the

number of servers per site for every delay limit according to a uniform distribution between 100 and 150. As illustrated in Fig. 7, for both SAO and SBO, as the delay limit increases, the cost decreases. The reason is that with higher delay limit more users meet delay requirement in each edge site, leading to more opportunities of resource sharing and cost reduction.

Fig. 8 shows the cost reduction ratio when τ is 40 ms. Different from the case when τ is 30 ms, SAO-I performs poorly now. With higher delay limit, more users meet delay requirement in each edge site. As a result, after the user assignment, more users may aggregate together. Packing in the granularity of instance causes resource waste due to large item size. As shown in Fig. 8, our algorithm SAO learns the best performance. As listed in the third line in Table 2, with 4000 instances, over 20 random simulations, the GD algorithm converges to the level of group in all cases.

6.3 Effect of θ

We evaluate the effect of θ on the performance, which is the weight of $|A|$ (the cardinality of partition parts) relative to c_v (server cost) in the set cost definition (23). With large θ , the user assignment algorithm prefers the partition parts which contain more users. Fig. 9 verifies this argument. It shows the empirical CDF for the size of partition parts resulted from 2000 instances.

Fig. 10 shows the cost reduction ratio of SAO under various θ values. For large scale problems, as θ increases from zero to one, the performance becomes better, because the weight of $|A|$ improves user aggregation in sites and resource sharing among them. However, as θ further increases, the performance becomes worse. This is because with the heavy weight on $|A|$, users are assigned to some edge site with high server cost. We set θ to one for a good balance between server cost and user aggregation.

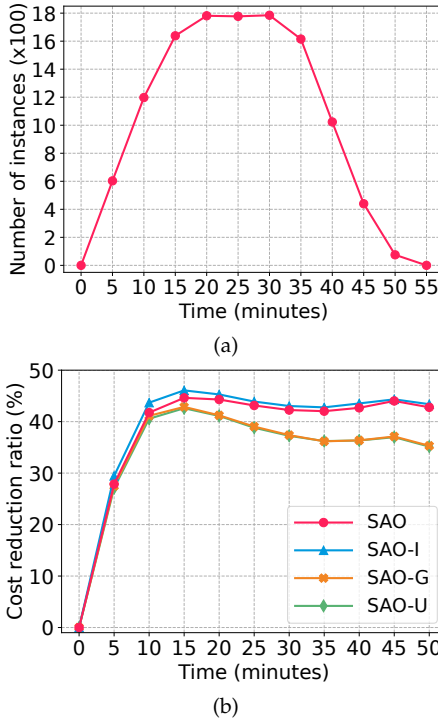


Fig. 11: The performance in a dynamic environment. (a) the number of active instances as the time goes; (b) the cost reduction ratio compared to SBO as the time evolves.

6.4 Performance in Dynamic Environment

When an instance starts, our algorithm assigns resources for it; when it finishes, the occupied resources will be released and reallocated later. We evaluate the performance of our algorithm in a dynamic environment, where 4000 instances arrive and depart sequentially. The inter-arrival time follows an exponential distribution with the mean of 500 milliseconds. The running time of instances follows a uniform distribution between 10 and 20 minutes. The other parameters are set as in Table 1. Our result is averaged over 20 random simulations. Fig. 11a shows the average number of running instances in the system as the time goes. For each round of simulation, the number of running instances increases first as instances arrive sequentially; then roughly speaking it reaches a maximum level just before the first departure and maintains this level for a while; finally it decreases after the last arrival. Fig. 11b shows the average cost reduction ratio as the time goes. It is observed that our algorithm (SAO) gets over 40% cost reduction compared to SBO and learns the best granularity for server packing.

6.5 One-User-Per-Instance Case

For the one-user-per-instance case, we compare the transportation-based offline offloading algorithm (TFO) proposed in Section 5 with the SBO algorithm. In TFO-GLPK, we use GLPK [15] to solve the relaxed transportation problem (30) and get its optimal solution in each iteration. In TFO-Vogel, we use Vogel's method [14] to solve the TP problem and get its feasible solution in each iteration. In this simulation, we use 15 edge sites, each containing 10 servers. The resource capacity p^U of a server is set to 20. As shown in Fig. 12a, both algorithms perform similarly, and can reduce cost by up to 11% when there are 1000 instances

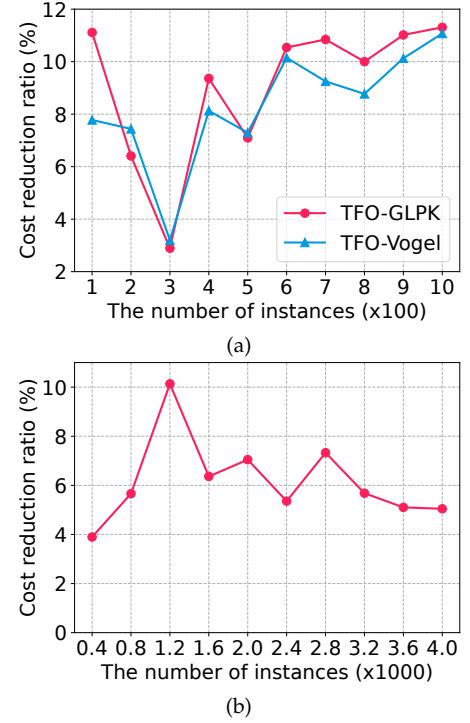


Fig. 12: The cost reduction ratio of the TFO algorithm for the one-user-per-instance case. (a) for small-scale problems; (b) TFO-Vogel for large-scale problems.

to offload. In Fig. 12b, we present the result of TFO-Vogel for large-scale problems where there are 30 sites, the number of servers per site follows a uniform distribution between 10 and 20, and there are 400~4000 instances. The cost reduction ratio is between 4%~10%. In the above simulation, the TFO algorithm takes 2 to 6 iterations to converge. It demonstrates that our algorithm is scalable for large scale problems.

7 RELATED WORK

Task offloading is an actively researched topic in edge computing. We review the existing related works and classified them into three categories in terms of optimization objectives: minimizing task delay, minimizing system cost and minimizing both.

First, some works aim to minimize the delay of tasks [16]–[25]. Chen *et al.* in [16] investigated a task offloading problem in mobile edge computing to minimize task duration while saving the battery life of user equipment. Chen *et al.* in [17] investigated a computation peer offloading problem in MEC-enabled small cell networks. It minimizes latency while considering limited energy resources. Yu *et al.* in [18] designed a collaborative offloading scheme with sharing computation results among tasks to minimize the execution delay for mobile users. Dai *et al.* in [19] investigated offloading tasks from vehicles to road side units. They jointly optimized load balancing and offloading to maximize system utility, which is a function of task processing delay. Alameddine *et al.* in [20] investigated a dynamic task offloading and scheduling problem which jointly optimizes task offloading, resource allocation and task scheduling. It maximizes the number of tasks meeting latency requirements. Tang *et al.* in [21] investigated the task offloading problem which determines the offloading decision for user

devices without knowing the uncertain load dynamics at edge nodes. They proposed a deep reinforcement learning algorithm to minimize the delay of a task and penalty if the task is dropped. Meng *et al.* in [22] studied online deadline-aware task dispatching and scheduling in edge computing to maximize the number of met deadlines. Wang *et al.* in [23] investigated the problem to offload dependent tasks with the minimum latency. They proposed a task offloading method based on meta reinforcement learning, which adapts fast to new environments with a small number of gradient updates and samples. Liu *et al.* in [24] proposed an adaptive task offloading algorithm for time-critical tasks. Liu *et al.* in [25] investigated a fog assisted cooperative service problem, which aims at minimizing the overall service delay. Different from this category of works, we consider the delay of tasks in constraints and minimize the resource consumption of edge systems.

Second, some works aim to minimize the cost of edge systems [26]–[31]. Wang *et al.* in [26] investigated the edge resource allocation problem which minimizes total cost of operation, service quality, reconfiguration and migration. They formulated it into a mixed nonlinear optimization problem and proposed an online algorithm without knowing user mobility pattern. Wang *et al.* in [27] proposed to minimize the energy consumption of MEC servers by jointly designing offloading strategy and sleep control scheme. Liu *et al.* in [28] investigated the task offloading problem in vehicular edge computing networks, where vehicles are edge nodes. It maximizes the total utility including communication cost and computation cost. They proposed deep reinforcement learning-based methods. Pu *et al.* in [29] investigated the task offloading problem to minimize the energy consumption of vehicles serving crowdsensing applications. They proposed an Lyapunov-based online task scheduling algorithm. Zhou *et al.* in [30] proposed an online orchestration framework for cross-edge service function chaining to minimize the holistic cost of edge system. He *et al.* in [31] investigated the problem to allocate app users to edge servers. It maximizes the number of app users to be served with minimum overall system cost. They proposed a game-theoretic approach and a decentralized algorithm. Different from these works, our research work consider the multi-dimensional resource sharing feature of tasks and leverage it to reduce resource consumption.

Third, some works minimize both task delay and system cost [32], [33]. Dink *et al.* in [32] investigated the task offloading problem from a mobile device to multiple access points. It minimizes both tasks' execution delay and the mobile device's energy consumption by jointly optimizing task allocation decision and the mobile device's CPU frequency. Yan *et al.* in [33] investigated offloading an application composed of a task call graph from a mobile device to some access points. It jointly optimizes offloading decision and resource allocation to minimize energy-time cost. These works do not consider the resource sharing among tasks and thus cannot achieve the least cost for the rendering tasks of interactive applications.

8 FUTURE WORKS

During the running of rendering tasks, the network conditions (*i.e.*, delay, bandwidth) between users and edge servers

may change, as a result the original allocation may not be the optimal any more. In addition, when some instances finish and their occupied resources are released, rearranging the active instances may consolidate resources and save cost. However, the rearrangement involves the live migration of running tasks. Long migration time would result in the interruption of the rendering service and poor user experience. Thus, the migration time must be considered in the rearrangement of rendering tasks. This complicates the offloading problem and is left for future work.

9 CONCLUSION

In this paper, we investigate a task offloading problem, that is where (on which server in which edge site) to offload rendering tasks such that each user will experience tolerable delay and meanwhile the cost of used servers is minimized. We leverage the multi-dimensional resource sharing feature of rendering tasks and propose a sharing-aware online task offloading algorithm so as to reduce cost. Our simulation results demonstrate that the sharing-aware algorithm reduces cost significantly compared to the sharing-oblivious one. In addition, we also discuss the special case of one-user-per-instance and propose a transportation-based offline offloading algorithm. Our simulation results demonstrate that this iterative algorithm can reduce cost compared to the greedy algorithm.

ACKNOWLEDGMENT

This work was supported by the China NSFC (61802263, 62072317), the grant from Research Grants Council of Hong Kong (CityU 11202419), the Faculty Research Fund of Shenzhen University (860/000002110325), the China NSFC (U2001207, 61872248), Guangdong NSF 2017A030312008, Shenzhen Science and Technology Foundation (No. ZDSYS20190902092853047, R2020A045), the Project of DEGP (No. 2019KCXTD005, 2021ZDZX1068), the Guangdong "Pearl River Talent Recruitment Program" under Grant 2019ZT08X603. Kaishun Wu is the corresponding author.

REFERENCES

- [1] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.
- [2] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1584–1607, 2019.
- [3] State of the edge 2020: A market and ecosystem report for edge computing. [Online]. Available: <https://stateoftheedge.com/reports/state-of-the-edge-2020>
- [4] R. Shea, J. Liu, E. C. . Ngai, and Y. Cui, "Cloud gaming: architecture and performance," *IEEE Network*, vol. 27, no. 4, pp. 16–21, 2013.
- [5] OnLive. [Online]. Available: <https://en.wikipedia.org/wiki/OnLive>
- [6] GeForce Now. [Online]. Available: <https://www.nvidia.com/en-us/geforce-now/>
- [7] CloudXR. [Online]. Available: <https://www.nvidia.com/en-us/design-visualization/solutions/cloud-xr/>
- [8] X. Zhang, H. Chen, Y. Zhao, Z. Ma, Y. Xu, H. Huang, H. Yin, and D. O. Wu, "Improving cloud gaming experience through mobile edge computing," *IEEE Wireless Communications*, vol. 26, no. 4, pp. 178–183, 2019.

- [9] S. Sukhmani, M. Sadeghi, M. Erol-Kantarci, and A. El Saddik, "Edge caching and computing in 5G for mobile AR/VR and tactile internet," *IEEE MultiMedia*, vol. 26, no. 1, pp. 21–30, 2019.
- [10] X. Liao, L. Lin, G. Tan, H. Jin, X. Yang, W. Zhang, and B. Li, "Liverender: A cloud gaming system based on compressed graphics streaming," *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 2128–2139, 2016.
- [11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [12] L. Babel, B. Chen, H. Kellerer, and V. Kotov, "On-line algorithms for cardinality constrained bin packing problems," in *Algorithms and Computation*, P. Eades and T. Takaoka, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 695–706.
- [13] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. USA: Prentice-Hall, Inc., 1982.
- [14] P. Gupta and D. Hira, *Operations Research*. S. Chand, 1992.
- [15] GLPK. [Online]. Available: <https://www.gnu.org/software/glpk/>
- [16] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.
- [17] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.
- [18] S. Yu, R. Langar, X. Fu, L. Wang, and Z. Han, "Computation offloading with data caching enhancement for mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 11 098–11 112, 2018.
- [19] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4377–4387, 2019.
- [20] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 668–682, 2019.
- [21] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Transactions on Mobile Computing*, pp. 1–12, 2020.
- [22] J. Meng, H. Tan, X.-Y. Li, Z. Han, and B. Li, "Online deadline-aware task dispatching and scheduling in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1270–1286, 2020.
- [23] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2021.
- [24] C. Liu, K. Liu, S. Guo, R. Xie, V. C. S. Lee, and S. H. Son, "Adaptive offloading for time-critical tasks in heterogeneous internet of vehicles," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 7999–8011, 2020.
- [25] K. Liu, K. Xiao, P. Dai, V. C. Lee, S. Guo, and J. Cao, "Fog computing empowered data dissemination in software defined heterogeneous vanets," *IEEE Transactions on Mobile Computing*, vol. 20, no. 11, pp. 3181–3193, 2021.
- [26] L. Wang, L. Jiao, J. Li, J. Gedeon, and M. Mhlhuser, "MOERA: Mobility-agnostic online resource allocation for edge computing," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1843–1856, 2019.
- [27] S. Wang, X. Zhang, Z. Yan, and W. Wenbo, "Cooperative edge computing with sleep control under nonuniform traffic in mobile edge networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4295–4306, 2019.
- [28] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 11, pp. 11 158–11 168, 2019.
- [29] L. Pu, X. Chen, G. Mao, Q. Xie, and J. Xu, "Chimera: An energy-efficient and deadline-aware hybrid edge computing framework for vehicular crowdsensing applications," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 84–99, 2019.
- [30] Z. Zhou, Q. Wu, and X. Chen, "Online orchestration of cross-edge service function chaining for cost-efficient edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1866–1880, 2019.
- [31] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in

edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 3, pp. 515–529, 2020.

- [32] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.

- [33] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on Wireless Communications*, vol. 19, no. 8, pp. 5404–5419, 2020.



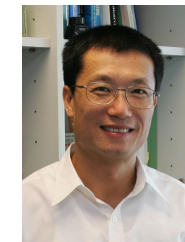
Ruitao Xie received her PhD degree in Computer Science from City University of Hong Kong in 2014, and BEng degree from Beijing University of Posts and Telecommunications in 2008. She is currently an assistant professor in College of Computer Science and Software Engineering, Shenzhen University. Her research interests include edge computing, AI networking, cloud computing and distributed systems.



Junhong Fang received a BEng degree in Computer Science from China University of Mining and Technology in 2020. He is currently a graduate student in College of Computer Science and Software Engineering, Shenzhen University.



Junmei Yao received a PhD degree in Computer Science from Hong Kong Polytechnic University in 2016, a MEng degree (2005) and a BEng degree (2003) from Harbin Institute of Technology, China. She is currently an assistant professor in the College of Computer Science and Software Engineering, Shenzhen University, China. Her research interests include wireless networks, wireless communications and mobile computing.



Xiaohua Jia received his BSc (1984) and MEng (1987) from University of Science and Technology of China, and DSc (1991) in Information Science from University of Tokyo. He is currently Chair Professor with Dept of Computer Science at City University of Hong Kong. His research interests include cloud computing and distributed systems, data security and privacy, computer networks and mobile computing. Prof. Jia is an editor of IEEE Internet of Things, IEEE Trans. on Parallel and Distributed Systems (2006–2009),

Wireless Networks, Journal of World Wide Web, Journal of Combinatorial Optimization, etc. He is the General Chair of ACM MobiHoc 2008, TPC Co-Chair of IEEE GlobeCom 2010-Ad Hoc and Sensor Networking Symp, Area-Chair of IEEE INFOCOM 2010, 2015–2017. He is Fellow of IEEE.



Kaishun Wu received his PhD degree in computer science and engineering from HKUST in 2011. After that, he worked as a research assistant professor at HKUST. In 2013, he joined SZU as a distinguished professor. He has co-authored 2 books and published over 100 high quality research papers in international leading journals and primer conferences, like IEEE TMC, IEEE TPDS, ACM MobiCom, IEEE INFOCOM. He is the inventor of 6 US and over 90 Chinese pending patent. He received 2012 Hong Kong

Young Scientist Award, 2014 Hong Kong ICT Awards: Best Innovation and 2014 IEEE ComSoc Asia-Pacific Outstanding Young Researcher Award. He is an IET Fellow.